EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Maestría

Arquitectura de servicios para la localización de personas y objetos en espacios cerrados

presentada por

Erick Infante Covarrubias

como requisito para la obtención del grado de

Maestro en Ciencias de la Computación

Director de tesis

Dr. Javier Ortiz Hernández

Codirectora de tesis

Dra. María Yasmín Hernández Pérez

Cuernavaca, Morelos, México. Febrero de 2023

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

Cuernavaca, Mor., 09/febrero/2023
No. De Oficio: SAC/043/2023
Asunto: Autorización de impresión de tesis

ERICK INFANTE COVARRUBIAS
CANDIDATO AL GRADO DE MAESTRO EN CIENCIAS DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"ARQUITECTURA DE SERVICIOS PARA LA LOCALIZACIÓN DE PERSONAS Y OBJETOS EN ESPACIOS CERRADOS",** ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

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

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

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SUBDIRECCIÓN ACADÉMICA

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO

C. c. p. Departamento de Ciencias Computacionales
Departamento de Servicios Escolares

CMAZ/RMA

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
09 FEB 2023
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
DEPTO. DE SERVICIOS ESCOLARES
RECIBIDO

Oficio de aceptación de documento de tesis

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO

Centro Nacional de Investigación y Desarrollo Tecnológico
Departamento de Ciencias Computacionales

Cuernavaca, Morelos, 03/febrero/2023
No. de Oficio: DCC/034/2023
Asunto: Aceptación de documento de tesis
CENIDET-AC-004-M14 OFICIO

JUAN GABRIEL GONZÁLEZ SERNA
JEFE DEL DEPARTAMENTO DE CIENCIAS COMPUTACIONALES
PRESENTE

Por este conducto, los integrantes de Comité Tutorial de ERICK INFANTE COVARRUBIAS, con número de control M19CE058, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis de grado titulado **"ARQUITECTURA DE SERVICIOS PARA LA LOCALIZACIÓN DE PERSONAS Y OBJETOS EN ESPACIOS CERRADOS"**, y hemos encontrado que se han atendido todas las observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

JAVIER ORTIZ HERNÁNDEZ
Director de tesis

MARÍA YASMÍN HERNÁNDEZ PÉREZ
Codirectora de tesis

ALICIA MARTÍNEZ REBOLLAR
Revisor 1

JOAQUÍN PÉREZ ORTEGA
Revisor 2

c.c.p. Archivo

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
09 FEB 2023
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

Oficio de autorización de impresión de tesis

Dedicatoria

"El mundo. ¡Qué lugar tan glorioso! Busca la libertad, y ella se extenderá ante tus ojos… si el sueño sin fin guía tu espíritu inquieto, ¡aprovéchalo! ¡Levanta tu bandera y mantente erguido!"

- Eiichirō Oda (Gold D. Roger)

A ti mi estimado lector, persigue tus sueños, aprovecha esa energía y no te rindas.

Este trabajo de investigación va dedicado a todas las personas que me han apoyado a lo largo de la vida, a mis padres Regino y Esmeralda y a mis hermanos Regino y Everardo.

Agradecimientos

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT) por la beca otorgada para la realización de mis estudios de maestría. Al Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), que forma parte del Tecnológico Nacional de México, por brindarme la oportunidad de convivir con grandes investigadores y compañeros.

Agradezco también a mi director de tesis Dr. Javier Ortiz Hernández, a mi codirectora de tesis Dra. María Yasmín Hernández Pérez y a los miembros del comité tutorial por su tiempo, conocimiento, dedicación, consejos y observaciones para mejorar esta investigación: Dra. Alicia Martínez y Dr. Joaquín Pérez.

A mi familia por siempre motivarme a seguir adelante y no rendirme. A mis padres Regino y Esmeralda por ser mi fuente de inspiración para seguir estudiando y por su amor incondicional. A mis hermanos Regino y Everardo por siempre competir y a su vez impulsarnos a ser mejores cada vez.

A Karla Lorena Cornejo Ramírez, por apoyarme siempre, por impulsarme y motivarme cada día para terminar mis proyectos, por su comprensión y apoyo emocional, por sus locuras que me alegran los días y su personalidad alegre. ¡Gracias!

Resumen

Las tecnologías de la información y la comunicación se han consolidado como un factor clave en la mejora de la calidad de vida y a su vez es un apoyo fundamental para las actividades diarias de las personas y organizaciones. Continuamente brindan nuevas aplicaciones y servicios en diversos ámbitos de la vida diaria: productividad, comunicación, seguridad, salud, entre otros. Muchas de estas aplicaciones permiten que las personas y objetos puedan estar conectados en tiempo real, intercambien información y nos auxilien en nuestras tareas cotidianas.

Un problema de interés práctico de que ha suscitado el estudio de diversas alternativas de solución es el posicionamiento de personas y objetos en espacios cerrados. Es decir al interior de edificios, como pueden ser centros comerciales, hospitales, escuelas, entre otros más, con la finalidad de guiarles para llegar a un destino, establecer polígonos de seguridad, detectar la proximidad a sitios peligrosos o zonas restringidas, realizar estudios de movilidad, etc. Es importante resaltar que las personas permanecen en promedio 87% del tiempo en espacios cerrados y estos cada vez son más grandes y más concurridos.

En esta tesis se propone una arquitectura de servicios que permita acoplar de una manera sencilla los componentes modulares de una aplicación de localización en espacios cerrados. La idea es que los componentes sean totalmente configurables, se puedan añadir, modificar o reemplazar de manera ágil sin alterar el funcionamiento de los servicios que se proporcionan.

Para el diseño de la arquitectura se optó por un diseño compuesto utilizando el patrón en capas como base para interactuar con distintas plataformas de IoT. Se realizó una implementación que interactúa con las plataformas FIWARE y DeviceHive. Se evaluó su desempeño utilizando una métrica específica para medir la modificabilidad de una arquitectura de software obteniendo resultados satisfactorios.

Abstract

Information and communication technologies have established themselves as a key factor in improving the quality of life and in turn is a fundamental support for the daily activities of people and organizations. They continuously provide new applications and services in various areas of daily life: productivity, communication, security, health, among others. Many of these applications allow people and objects to be connected in real time, exchange information and help us with our daily tasks.

A problem of practical interest that has raised the study of various solution alternatives is the positioning of people and objects in closed spaces. That is to say, inside buildings, such as shopping centers, hospitals, schools, among others, in order to guide them to reach a destination, establish security polygons, detect proximity to dangerous sites or restricted areas, carry out security studies. mobility etc It is important to highlight that people spend an average of 87% of the time in closed spaces and these are getting bigger and more crowded.

In this thesis, a service architecture is proposed that allows the simple coupling of the modular components of a localization application in closed spaces. The idea is that the components are fully configurable, they can be added, modified or replaced in an agile way without altering the operation of the services that are provided.

For the design of the architecture, a composite design was chosen using the layered pattern as a basis to interact with different IoT platforms. An implementation was made that interacts with the FIWARE and DeviceHive platforms. Its performance was evaluated using a specific metric to measure the modifiability of a software architecture obtaining satisfactory results.

Tabla de contenido

Resumen 1
Abstract 2
Tabla de contenido 3
Lista de figuras 7
Lista de tablas 8
Capítulo 1 9
1. Introducción 10
1.1. Contexto de la investigación 10
1.2. Planteamiento del problema 11
1.3. Objetivo general 13
1.4. Objetivos específicos 13
1.5. Alcances y limitaciones de la investigación 13
1.6. Organización del documento 14
Capítulo 2 15
2. Marco teórico 16
2.1. Arquitectura de software 16
2.1.1. Patrones de la arquitectura 17
2.2. Arquitectura de software de IoT 24
2.3. Arquitectura de software de IoT orientada a servicios 25
2.4. Requisitos a considerar en la arquitectura 26
2.5. Componentes y servicios 26
2.6. FIWARE 27
2.7. DeviceHive 28
2.8. Comparación de las plataformas FIWARE y DeviceHive 29
2.8.1. Confiabilidad 30
2.8.2. Escalabilidad 31
2.8.3. Heterogeneidad 31
2.8.4. Usabilidad 32
2.9. Servicios Web 33
2.10. Interfaz de programación de aplicaciones 33
2.11. Edificios inteligentes 33
2.12. Geolocalización 34

2.12.1. Localización en espacios cerrados 35
2.13. Tecnologías emisoras de señales inalámbricas 35
2.13.1. WiFi 35
2.13.2. Bluetooth de baja energía 35
Capítulo 3 37
3. Revisión de la literatura 38
3.1. Antecedentes en CENIDET 38
3.2. Revisión de la literatura 40
3.2.1. Rendimiento y desafíos de la arquitectura orientada a servicios para redes de sensores inalámbricos 41
3.2.2. Un enfoque cuantitativo para analizar la modificabilidad en el diseño arquitectónico de software de sistemas de aplicaciones ágiles 42
3.2.3. Investigación sobre la arquitectura general del middleware de Internet de las cosas para parques industriales inteligentes 45
3.2.4. Soluciones de Amazon, Google y Microsoft para IoT: arquitecturas y una comparación de rendimiento 46
3.2.5. Internet de las cosas: un estudio sobre arquitectura, tecnologías, protocolos y desafíos 48
3.2.6. LISA: arquitectura ligera de servicios de IoT sensible al contexto 49
3.2.7. Un marco ligero para soluciones inteligentes de IoT 50
3.2.8. Comparación de la localización en interiores basada en la intensidad de señales para edificios inteligentes con internet de las cosas 52
3.2.9. Un breve estudio de conceptos de arquitectura de software y arquitectura orientada a servicios 53
3.2.10. Un modelo de identificación de servicios para la arquitectura orientada a servicios 54
3.2.11. Estado de la práctica 55
3.2.11.1. Situm 55
3.2.11.2. Mapwize 56
3.2.11.3. Indoora 57
3.2.11.4. Navisens 57
3.2.11.5. Litum IoT 58
3.3. Observaciones de trabajos relacionados 59
3.3.1. Arquitecturas 59
3.3.2. Tecnologías de localización en interiores 59
3.3.3. Evaluación de requerimientos 60
3.4. Comparación de trabajos relacionados 60
3.4.1. Arquitecturas 60

3.4.2. Tecnologías 61
3.4.3. Evaluación de Requerimientos 62
Capítulo 4 63
4. Diseño y desarrollo de la arquitectura 64
4.1. Diseño de la arquitectura 64
4.1.1. Componentes principales 66
4.2. Implementación de la arquitectura 68
4.2.1. Implementación de la capa de servicios de usuario 68
4.2.2. Implementación de la capa de IoT 73
4.2.3. Implementación de la capa de posicionamiento 73
4.2.4. Implementación de la capa de dispositivo 74
4.3. Mapeo FIWARE y DeviceHive 74
4.4. Flujo de datos 75
4.4.1. Modelo de datos NGSI 75
4.4.2. Capa de dispositivo 78
4.4.3. Capa de posicionamiento 79
4.4.4. Capa de IoT 79
4.4.5. Capa de servicios de usuario 79
4.5. Modificabilidad en los servicios 79
4.5.1. Servicios de usuario (WEB) 80
4.5.2. Servicios de IoT 80
4.5.3. Servicios de posicionamiento 81
Capítulo 5 83
5. Caso de estudio 84
5.1. Construcción del caso de estudio 85
5.1.1. Escenario de pruebas 86
5.2. Caso de estudio enfocado a FIWARE 87
5.3. Caso de estudio enfocado a DeviceHive 88
5.4. Caso de estudio con ambas plataformas 89
Capítulo 6 91
6. Plan de pruebas 92
6.1. Evaluación de la arquitectura 93
6.1.1. Evaluación de la arquitectura usando FIWARE 94
6.1.2. Evaluación de la arquitectura usando DeviceHive 95

6.1.3. Evaluación de la arquitectura usando FIWARE y DeviceHive ... 96
6.2. Comparación de resultados ... 97
Capítulo 7 ... 98
7. Conclusiones y trabajo futuro ... 99
7.1. Conclusiones ... 99
7.2. Trabajos futuros ... 100
Referencias ... 102
Anexo A ... 106
Productos académicos ... 106
Anexo B ... 109
Requerimientos ... 111
Instalación ... 113
Instalación de Ubuntu Server 22 ... 113
Actualizar e instalar componentes adicionales ... 116
Instalar Docker-compose ... 118
Despliegue de FIWARE ... 118
Despliegue de DeviceHive ... 119

Lista de figuras

Figura 1 Arquitectura en capas 17
Figura 2 Arquitectura de broker 18
Figura 3 Arquitectura Modelo-Vista-Controlador 19
Figura 4 Arquitectura de Tubería y Filtro 20
Figura 5 Arquitectura Cliente Servidor 21
Figura 6 Arquitectura Punto a Punto 22
Figura 7 Arquitectura publicación-suscripción 22
Figura 8 Arquitectura Orientada a Servicios 23
Figura 9 Arquitectura de 3 capas (a) y 5 capas (b) 24
Figura 10 Diseño de la arquitectura 65
Figura 11 Interfaz web con dos servicios 69
Figura 12 Archivo HTML para interfaz web 69
Figura 13 Servicio de mapa 70
Figura 14 Fragmento de código PHP para servicio de visualización 71
Figura 15 Servicio de alertas 72
Figura 16 Fragmento de código PHP para servicio de alerta 72
Figura 17 Fragmento de código Python para servicio de posicionamiento 74
Figura 18 Modelo de datos NGSI para FIWARE 77
Figura 19 Modelo de datos adaptado NGSI DeviceHive 78
Figura 20 Archivo YML de FIWARE 82
Figura 21 Escenario de pruebas 86
Figura 22 Arquitectura para caso de estudio de FIWARE 87
Figura 23 Arquitectura para caso de estudio de DeviceHive 88
Figura 24 Arquitectura para caso de estudio simultáneo 89
Figura 25 Configuración del servidor RAM 111
Figura 26 Configuración del servidor Disco Duro 112
Figura 27 Configuración del servidor Procesador 112
Figura 28 Instalar Ubuntu Server 113
Figura 29 Seleccionar idioma 113
Figura 30 Base de instalación completa 114
Figura 31 Resumen de configuración 115
Figura 32 Instalación de servidor SSH 115
Figura 33 Instalación de Docker 116
Figura 34 Actualización del sistema operativo 117
Figura 35 Bitvise SSH 117
Figura 36 Resultado de ejecución 118
Figura 37 Fiware en funcionamiento 119
Figura 38 Procesos Docker 119
Figura 39 DeviceHive en funcionamiento 120
Figura 40 Procesos de DeviceHive 120

Lista de tablas

Tabla 1. Valores de Modificabilidad en una arquitectura 44
Tabla 2 Comparación de trabajos sobre Arquitecturas de Software 61
Tabla 3 Comparación de trabajos sobre tecnologías inalámbricas 61
Tabla 4 Comparación de trabajos sobre evaluación de requerimientos 62
Tabla 5 Métrica de Modificabilidad 93
Tabla 6 Variables FIWARE 94
Tabla 7 Variables DeviceHive 95
Tabla 8 Variables del caso de estudio 96
Tabla 9 Resumen de resultados 97

Capítulo 1

Introducción

1. Introducción

La tecnología es un factor clave en la mejora de la calidad de vida y un apoyo fundamental para las actividades diarias de las personas y las organizaciones. Tiene áreas de aplicación y ofrece servicios para incrementar la productividad, mejorar la comunicación, lograr un mejor rendimiento, así como el aprovechamiento del tiempo.

Para ello se busca el diseño y desarrollo de herramientas que permitan que personas y objetos puedan estar conectados en tiempo real, intercambien información y nos auxilien en nuestras tareas cotidianas. El desarrollo de herramientas de esta naturaleza ha dado lugar al Internet de las Cosas (IoT, del inglés *Internet of Things*).

1.1. Contexto de la investigación

Las tecnologías de la información y la comunicación se han consolidado como un factor clave en la mejora de la calidad de vida y a su vez es un apoyo fundamental para las actividades diarias de las personas y organizaciones. Estas brindan continuamente nuevas aplicaciones y servicios en diversos ámbitos de la vida diaria, en particular para incrementar la productividad, mejorar la comunicación y el aprovechamiento del tiempo.

Para ello se busca el diseño y desarrollo de herramientas que permitan que las personas y objetos puedan estar conectados en tiempo real, intercambien información y nos auxilien en nuestras tareas cotidianas. El desarrollo de herramientas de esta naturaleza ha dado lugar al Internet de las Cosas (IoT, del inglés *Internet of Things*).

Un problema que concierne al IoT es el posicionamiento de una persona o de objetos que se encuentran en espacios cerrados, es decir al interior de edificios, como pueden ser centros comerciales, hospitales, escuelas, entre otros más, con la finalidad de guiarles para llegar a un destino, establecer polígonos de seguridad, detectar la proximidad a sitios peligrosos o zonas restringidas, realizar estudios de movilidad, etc. Es importante resaltar que las personas permanecen en promedio 87% del tiempo en espacios cerrados [1] y estos cada vez son más grandes y concurridos, dificultando cada vez más la localización e identificación de puntos de interés en el entorno.

En la actualidad existen tecnologías con la capacidad de localizar personas y objetos en espacios cerrados con una margen de incertidumbre de tres metros aproximadamente. Para ello, existe una variedad de tecnologías disponibles, tales como la banda ultra-ancha (UWB,

Ultra-Wide Band, por sus siglas en inglés), la cual soluciona problemas de localización en interiores con una mayor cobertura y precisión sin la necesidad de estar directamente en la línea de visión del dispositivo [2]. Los celulares inteligentes con acceso a una red de área local inalámbrica (WLAN, del inglés *Wireless Local Area Network*) son fácilmente localizables dentro de edificios. Por un lado, la tecnología WiFi está diseñada para conexiones de largo alcance, pero los dispositivos requieren una mayor cantidad de energía para enviar y recibir las señales ya que consume en promedio alrededor de 100 a 350 miliamperios [3]. Por otro lado, la tecnología *Bluetooth* está diseñada para dispositivos portátiles con un suministro de energía limitado con un consumo promedio de energía se encuentra entre 1 a 35 miliamperios.

Se consideró que una combinación de estas tecnologías (*WiFi* y *Bluetooth*) en conjunto con una colocación y configuración eficiente de los equipos emisores podría permitir localizar dispositivos móviles con una mayor precisión y a su vez reducir el consumo de energía de estos dispositivos.

De lo anterior se desprende la viabilidad del desarrollo de una arquitectura de servicios que integre las mejores alternativas tecnológicas, así como también las mejores prácticas y experiencias del estado del arte, con la posibilidad de ofrecer diversos servicios en espacios cerrados, tales como:

- Posicionamiento en tiempo semi real (5-10 segundos) en mapa bidireccional.
- Alarmas inteligentes para cuestiones de asistencia y accesibilidad de personas como alarmas de emergencia al salir de la zona de monitoreo.

El objetivo de una arquitectura de servicios es separar la lógica de integración de negocio de la implementación. Para lograrlo, se crean componentes que contienen la implementación de servicios individuales necesarios para los procesos de negocio [4]. Este tipo de solución permite explotar de una manera eficaz y eficiente la creación de nuevos servicios de localización en espacios cerrados, de manera ágil, en un menor tiempo y por lo tanto, con un menor costo de implementación.

1.2. Planteamiento del problema

Desde el año 2000, el estudio del comportamiento de los humanos en sociedad ha dado a la luz que las tendencias de la humanidad es convivir en espacios cerrados, todo esto se debe a diversos factores. De acuerdo con un estudio que busca observar los patrones de actividad de

las personas [1] mostro que las personas pasamos alrededor del 6% del tiempo dentro de un vehículo, 7% en espacios abiertos y el 87% en espacios cerrados, donde los servicios tradicionales de localización basados en GPS proporcionan un gran margen de incertidumbre.

Actualmente, el 55% de las personas viven en áreas urbanas y se estima [5] que para el año 2050 el 68% de la población mundial vivirá en las ciudades. Por lo tanto, el desarrollo de infraestructura y tecnologías que apoyen la accesibilidad, la vida activa y asistida son importantes, especialmente en el contexto del desarrollo de ciudades inteligentes [6].

En este contexto, la localización de personas en espacios cerrados es un tema que reserva mayor interés por las múltiples aplicaciones que podemos encontrar: trazado de rutas a personas en grandes espacios cerrados, establecimiento de polígonos de seguridad, estudios de marketing, etc.

En la literatura podemos encontrar diversas tecnologías que compiten para asegurar mayor precisión y confiabilidad, así como nuevas funcionalidades que sean de provecho para las personas [7]. La mayoría de estos calculan la ubicación a partir de la lectura de la intensidad de la señal de dispositivos emisores de WiFi, Beacon, RFID (del inglés *Radio Frequency Identification*) y NFC (del inglés *Near Field Communication*)[8]–[10].

Para el cálculo de la ubicación, los sensores de los teléfonos inteligentes toman como entrada una o varias señales inalámbricas emitidas por diversos dispositivos distribuidos dentro de las instalaciones, para después efectuar el cálculo de la ubicación utilizando uno de los muchos algoritmos de localización dependiendo de las necesidades que se buscan satisfacer con la aplicación.

Tomando en cuenta lo que se expone en los artículos anteriormente mencionados [8]–[10]para la localización en espacios cerrados, observamos que la utilización de dos o más tipos de señales, así como una cuidadosa combinación del uso de diversos algoritmos contribuye a obtener una localización con un menor margen de incertidumbre en la localización.

La problemática que se presenta es la necesidad de implementar o modificar servicios IoT en una plataforma de forma ágil, pero se han encontrado dificultades en el proceso debido a la existencia de soluciones generales que son complejas de administrar e implementar. Esto representa un desafío para lograr una mayor eficiencia en la implementación de servicios, y requiere ser abordado para lograr una implementación exitosa. Es importante encontrar una

solución para poder mejorar la capacidad de implementar o modificar servicios IoT de manera rápida y eficiente.

En esta tesis se propone una arquitectura de servicios que permita acoplar de una manera sencilla los componentes modulares de los servicios que integran una aplicación IoT. En nuestro caso, la arquitectura será utilizada para la implementación de un sistema de localización en espacios cerrados en la cual los componentes sean totalmente configurables, se puedan añadir, modificar o reemplazar de manera ágil sin alterar el funcionamiento de los servicios que se proporcionan.

1.3. Objetivo general

Desarrollar una arquitectura orientada a servicios enfocada en la localización de personas en espacios cerrados, capaz de soportar, crear, implementar, modificar y eliminar diversos servicios de manera ágil.

1.4. Objetivos específicos

1. Diseño de una arquitectura de IoT orientada a servicios que permita implementar, modificar y eliminar servicios de forma rápida y poco intrusiva al funcionamiento del resto de componentes.
2. Evaluar el diseño de la arquitectura de forma cuantitativa mediante métricas enfocadas en medir la modificabilidad presente en la arquitectura y la relación de las interacciones de sus componentes.
3. Diseñar e implementar un prototipo experimental que permita mostrar las funcionalidades de la arquitectura propuesta para servicios de localización y demostrar los beneficios que ofrece para su desarrollo y aplicación en diferentes escenarios.

1.5. Alcances y limitaciones de la investigación

Los alcances fueron los siguientes:

- Diseño de la arquitectura con base en patrones arquitectónicos comunes, pudiendo combinar estos mismos.
- Desarrollo de una arquitectura que facilite incorporar y desincorporar servicios de geoposicionamiento en espacios cerrados.

- Despliegue de la solución en dos plataformas para IoT: FIWARE y DeviceHive
- Evaluación de la arquitectura utilizando métricas específicas reportadas en la literatura.

Las limitaciones fueron las siguientes:

- Para propósitos del estudio solo se simularon los datos de posicionamiento de personas mediante una aplicación en Python que envíe los datos a la plataforma IoT.
- Las pruebas se realizaron con un teléfono inteligente con sistema operativo Android de gama media/alta.
- La experimentación y pruebas se realizó en un edificio habitacional utilizando la instalación de redes inalámbricas existentes.
- El prototipo solo funciona con componentes de la plataforma FIWARE y DeviceHive.
- Las características hardware de los servidores remotos están limitadas a la aplicación y a los servicios desarrollados.
- El prototipo solo consta de los siguientes servicios:
  - Visualización del dispositivo de las personas en un mapa bidimensional del edificio.
  - Envío de alertas de salida de las personas de la zona de monitoreo.

1.6. Organización del documento

Esta tesis consta de 8 Capítulos. En el Capítulo 2 se elabora una marco teórico en relación con las tecnologías existentes, los métodos y el funcionamiento general del posicionamiento en interiores. En el Capítulo 3 se describe el trabajo relacionado que fueron identificados para la presente investigación. En el Capítulo 4, se describe la arquitectura para dar solución a la problemática y los elementos principales que la constituyen. En el Capítulo 5 se plantea un caso de estudio para mostrar la arquitectura implementada. En el capítulo 6 se describen las pruebas y se evalúan los resultados de estas. Por último, en el Capítulo 7 se describen las conclusiones de la tesis y los trabajos futuros que se pueden realizar para ampliar esta investigación.

Capítulo 2

Marco Teórico

2. Marco teórico

En este capítulo, se explican los conceptos que se incluyen en el trabajo de investigación, haciendo énfasis en las arquitecturas de software y las plataformas IoT utilizadas.

2.1. Arquitectura de software

Una arquitectura de software es un conjunto de estructuras que son necesarias para entender un sistema, comprende los elementos de software, así como también las propiedades y relaciones entre ellos. Debido a este conjunto de estructuras, la arquitectura omite información que no es relevante para entender el sistema, o aquella información que no afecta el funcionamiento de otro elemento.

Los tipos de estructuras en una arquitectura de software son [11] :

- Módulos.- Son estructuras estáticas que se enfocan en las funcionalidades del sistema, los módulos permiten ser divididos de forma más eficiente entre los equipos encargados de su desarrollo. Este tipo de estructura nos permite responder a preguntas como: ¿Cuál es la función principal de este módulo? ¿Cuáles módulos dependen de otros?, entre otras.
- Componente y conector.- Es una estructura dinámica, el principal objetivo es mostrar la forma en que los elementos interactúan durante la ejecución del sistema. Este tipo de estructura nos permite responder preguntas como: ¿Cuáles son los principales componentes de la ejecución y cómo interactúan en tiempo de ejecución?, ¿Cuáles son los principales almacenes de datos compartidos?, ¿Cómo es el flujo de datos en el sistema?, entre otras.
- Asignación.- Son estructuras que muestran las relaciones entre los elementos de software con las entidades en las que son creados o ejecutados. Este tipo de estructura nos permite responder preguntas como: ¿En qué procesador se ejecuta cada elemento del software? ¿En qué directorio o archivo se encuentran almacenados cada elemento durante el desarrollo, pruebas y despliegue?

En los sistemas los elementos interactúan entre sí mediante interfaces que dividen los detalles de un elemento en su parte privada y pública. Una arquitectura se preocupa por el

lado público de los elementos, aquellos detalles que son de implementación interna no son arquitectónicos.

2.1.1. Patrones de la arquitectura

Un patrón arquitectónico establece una relación entre: un contexto recurrente, un problema y una solución. La descripción de un patrón arquitectónico describe el problema y sus posibles variantes, esta descripción incluye los atributos de calidad que deben cumplirse. La solución describe la estructura de la arquitectura que permite solventar el problema.

El uso de patrones arquitectónicos no limita el uso de uno solo, en la práctica estos se combinan para obtener un diseño que mejore los aspectos que eran deficientes de una arquitectura, reemplazándolos con otros más eficientes en ese aspecto. Sistemas complejos pueden mostrar múltiples patrones al mismo tiempo.

Los patrones pueden ser categorizados por los elementos que predominan en su diseño de acuerdo con [11] los patrones modulares son los siguientes:

2.1.1.1 Patrones modulares

Patrón en capas se utiliza para patrones complejos que necesitan ser desarrollados o modificados de forma independiente. El software necesita ser separado de tal forma que los módulos sean desarrollados y modificados con la mínima interacción con el resto de los componentes, otorgando así las características de portabilidad, reusabilidad y modificabilidad a dichos componentes del sistema.

Para lograr resolver las necesidades, el patrón en capas divide el software en unidades denominadas capas, donde cada capa es un agrupamiento de módulos que ofrecen servicios afines a su misma capa. Como limitante en este patrón las relaciones de uso son estrictamente de una capa superior o inferior, pudiendo existir en algunos casos un “puente” para conectar capas que no son inmediatas. La Figura 1 muestra un diagrama básico de la arquitectura en capas.

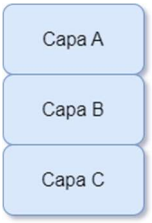

Figura 1 Arquitectura en capas

2.1.1.2 Patrones de componente – conector

El patrón de broker se utiliza en sistemas construidos de múltiples servicios distribuidos en múltiples servidores, por lo cual, la implementación de estos sistemas puede llegar a ser muy compleja si no se toma en cuenta la forma de comunicación y la disponibilidad de los servicios.

El patrón broker permite separar a los clientes (usuarios que consumen los servicios) de los proveedores de servicio mediante un intermediario denominado broker. El patrón funciona de la siguiente manera, cuando un usuario requiere un servicio, éste realiza la petición al Broker, y a su vez envía la solicitud al servidor correspondiente donde se encuentra el servicio requerido. Una vez procesada la información y realizando el proceso inverso, el broker regresa el resultado de la petición del usuario.

Gracias al sistema del broker, toda la información relacionada con los servidores queda protegida, y en el caso que algún servidor no esté disponible, el bróker tiene la capacidad de seleccionar un reemplazo que pueda cumplir con la petición del usuario. La limitante de este sistema es que, si no se implementa de la manera adecuada, el mismo bróker puede ser el punto de falla crítico en los sistemas más grandes y complejos, así como también, ocasionar cuellos de botella para la comunicación. La Figura 2 muestra un ejemplo de la arquitectura de broker.

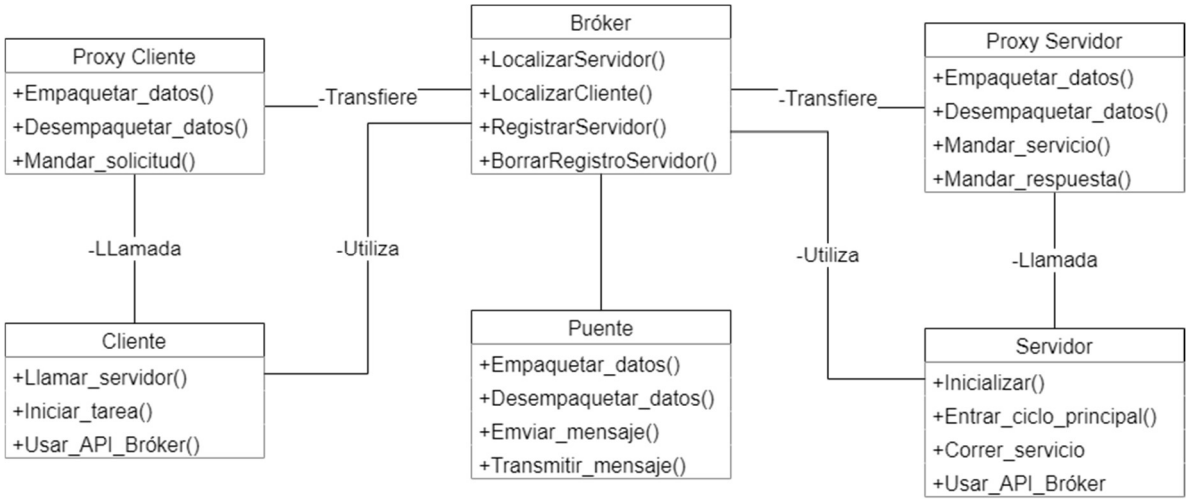

Figura 2 Arquitectura de broker

El patrón de Modelo-Vista-Controlador (MVC) es utilizado cuando la interfaz de software es la parte más modificada en aplicaciones interactivas. Por esta razón, es importante que las modificaciones de la interfaz sean realizadas de forma separada del resto del sistema.

Para implementar este patrón generalmente se presenta la pregunta ¿Cómo mantener la funcionalidad de la interfaz de manera separada y aun así ser responsiva?, esto se logra gracias a que el MVC separa la funcionalidad de la aplicación en tres tipos de componentes: El modelo, que contiene los datos de la aplicación, la Vista, que muestra una parte de los datos y permite la interacción con el usuario, y por último, el Controlador, que funciona como mediador entre el modelo y la vista y a su vez maneja las notificaciones de un cambio de estado a otro. La limitación de este patrón es que debe de existir al menos una instancia de cada modelo, vista y controlador. La Figura 3 muestra la arquitectura MVC y su comportamiento.

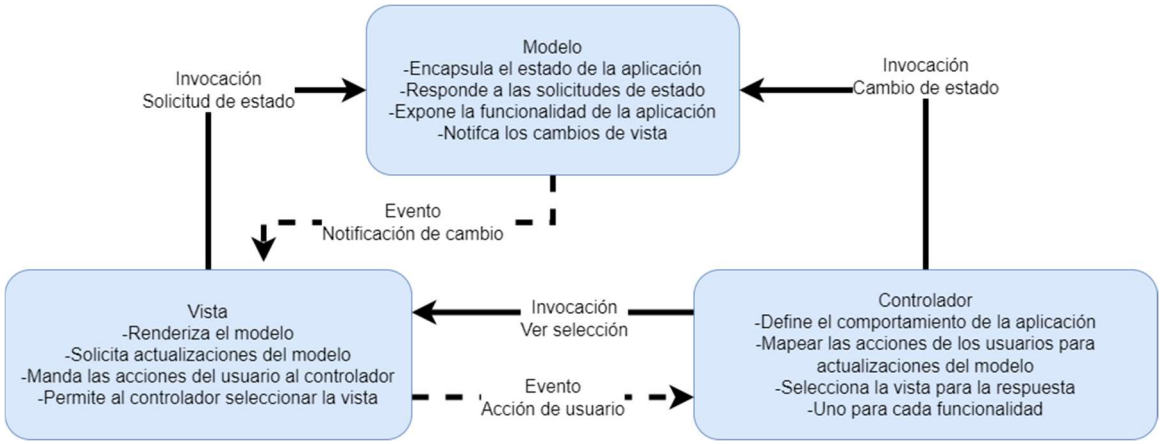

Figura 3 Arquitectura Modelo-Vista-Controlador

El patrón de Tubería y Filtro se utiliza en sistemas que requieren realizar procesos a un flujo continuo de datos, desde su entrada, hasta la salida. Generalmente es empleado cuando existen muchos tipos de transformaciones en los datos y ocurren en repetidas ocasiones durante el proceso, por lo cual es deseable desarrollarlos como partes independientes y reutilizables. Esta clase de sistemas necesita ser dividida en componentes que sean reusables y con poca o nula dependencia de otros componentes, y a su vez, que utilicen mecanismos de interacción simples y genéricos para que puedan ser reutilizados y puedan combinarse con otros existentes.

El patrón está caracterizado por las transformaciones sucesivas de flujos de datos. El flujo de información ingresa por una tubería, donde se aplica el proceso de filtrado o modificación, y se genera una salida, una característica de las tuberías es que pueden consumir o producir datos de una o más tuberías. La limitante de este patrón es que las tuberías conectadas deben de ser compatibles con las entradas y salidas de los datos, generalmente no es recomendable para sistemas interactivos. La Figura 4 muestra un ejemplo de organización de datos usando el patrón de tubería y filtros.

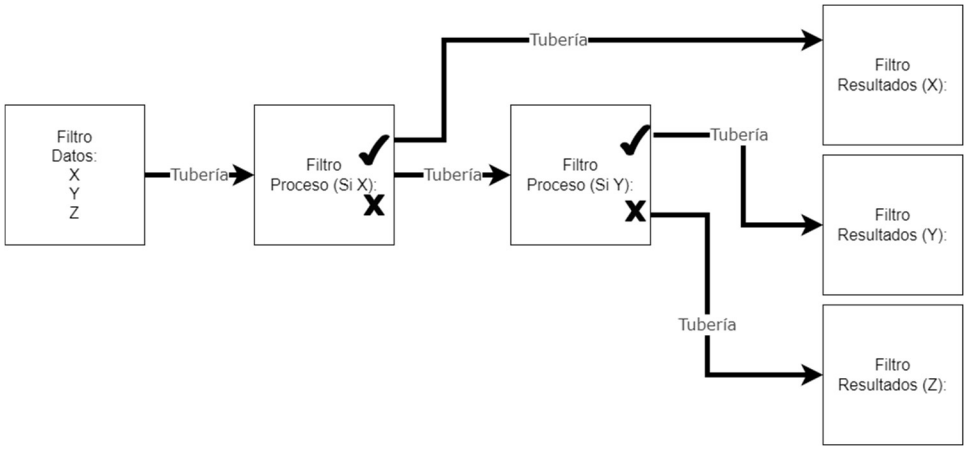

Figura 4 Arquitectura de Tubería y Filtro

El patrón Cliente-Servidor se utiliza cuando existen recursos y servicios compartidos a los que un gran número de clientes desean acceder y es necesario poder controlar el acceso y la calidad del servicio. El problema que se busca solucionar es la falta de escalabilidad y disponibilidad de los recursos.

La solución de este patrón es que los clientes solicitan servicios a los servidores encargados de proveerlos, donde algunos componentes pueden funcionar como clientes o servidores, y a su vez puede existir solo un servidor central o múltiples distribuidos. Las limitantes se presentan cuando solo es un servidor o múltiples configurados erróneamente, lo que puede ocasionar cuellos de botella para las conexiones. La Figura 5 muestra un diagrama de la arquitectura.

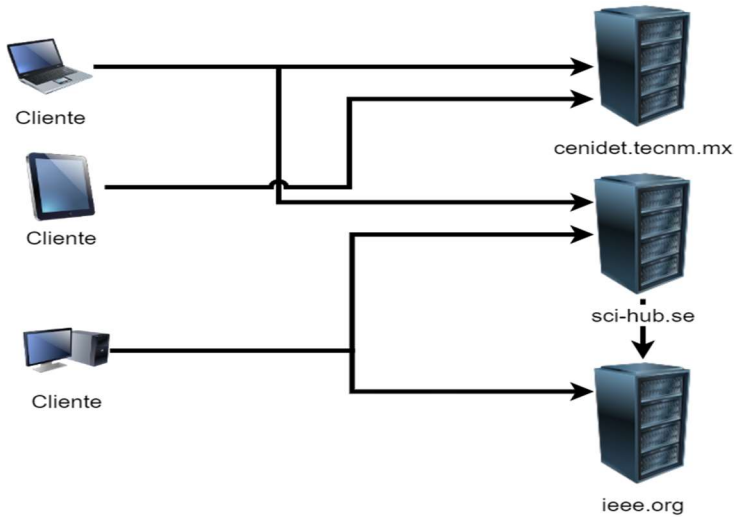

Figura 5 Arquitectura Cliente Servidor

El patrón de Punto a Punto es utilizado cuando existen unidades de cómputo distribuidas que necesitan colaborar para proveer servicios a una comunidad de usuarios distribuidos. De esta necesidad surge el problema de cómo realizar conexiones entre sí, por lo cual necesitan un protocolo de comunicación en común que les brinda alta disponibilidad y escalabilidad.

El diseño de cada componente es tratado como un "Punto", donde todos los puntos son iguales, y estos no pueden ser críticos para el comportamiento general del sistema. La comunicación punto a punto es generalmente una acción continua de solicitudes y respuestas hasta terminar la comunicación, las interacciones se pueden originar desde cualquier punto en interacciones por lo general bidireccionales, pudiendo ser en algunos casos solo envío de datos, sin requerir respuesta. Los puntos necesitan conectarse a la red de punto a punto para poder descubrir los otros puntos con los que pueden interactuar y así empezar a comunicarse para cumplir con su objetivo.

Los puntos pueden ser agregados o eliminados de la red y esto no genera un impacto significativo, lo cual resulta en una forma de escalabilidad para el sistema y proporciona flexibilidad para el desarrollo de sistemas a través de múltiples plataformas distribuidas. En cuestiones de limitantes que tiene este patrón son que en los casos donde la red sea muy extensa las cuestiones de seguridad, disponibilidad, respaldo y recuperación de datos o servicios puede ser muy compleja; por otro lado, en redes muy pequeñas existe la posibilidad de que las metas de desempeño y disponibilidad no se cumplan. La Figura 6 muestra un diagrama de la arquitectura Punto a Punto.

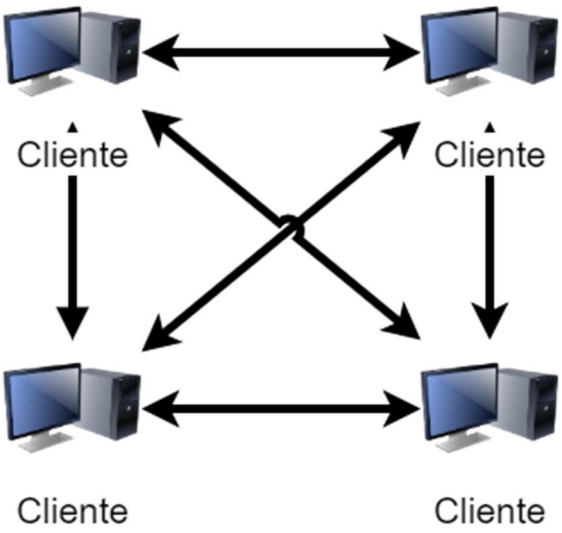

Figura 6 Arquitectura Punto a Punto

El patrón de publicación-suscripción se utiliza cuando existen múltiples productores y consumidores de datos que deben de interactuar para realizar los procesos. El problema que surge es la forma de implementar mecanismos para lograr una comunicación invisible entre productores y consumidores de servicios.

La solución es que los componentes interactúen mediante la publicación de mensajes o eventos. Los componentes pueden suscribirse a un conjunto de eventos, para después el bus de eventos se encargue de enviar a todos los componentes suscritos a cada evento, los componentes encargados de la publicación de eventos realizan un anuncio de evento al bus de eventos, este se encarga de enviarlo a todos los componentes que están interesados en el evento.

A mayor número de componentes en el sistema, el incremento en el tiempo de respuesta de peticiones puede llegar a afectar la escalabilidad, también puede verse afectado el control y envío de mensajes. La Figura 7 muestra un diagrama de la arquitectura.

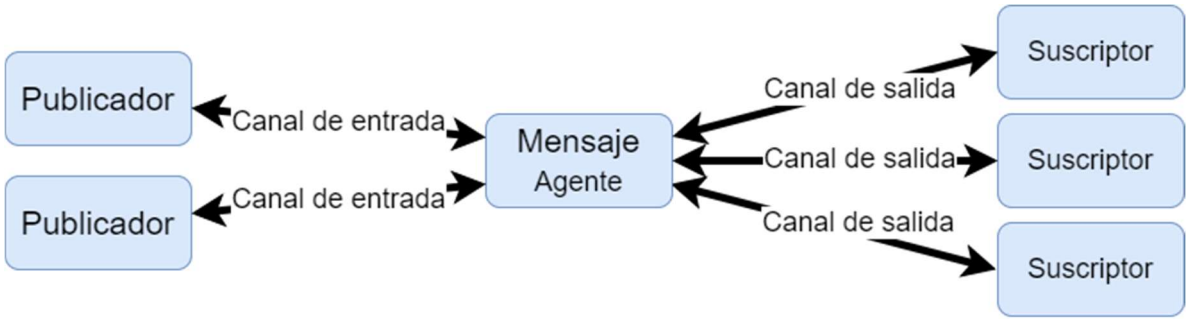

Figura 7 Arquitectura publicación-suscripción

El patrón de Arquitectura Orientada a Servicios es utilizado cuando existen proveedores de servicio que listan y describen sus servicios listos para ser consumidos por los clientes. En este contexto, los clientes deben de ser capaz de entender y usar estos servicios sin la necesidad de conocer la implementación de estos.

El patrón describe un conjunto de componentes distribuidos que pueden proporcionar o consumir servicios. En esta clase de sistemas tanto los componentes proveedores de servicio como los componentes consumidores de servicio pueden ser implementados en diversas plataformas y lenguajes, ofreciendo así un beneficio como la interoperabilidad de los componentes, ofreciendo los elementos necesarios para interactuar con servicios externos que se encuentran en la red. La Figura 8 muestra un diagrama de la arquitectura SOA.

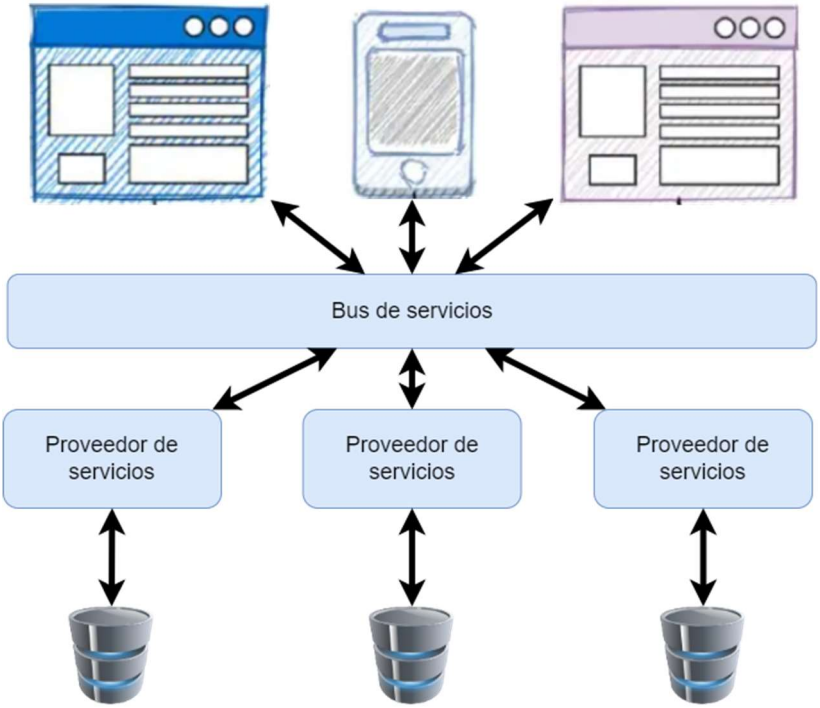

Figura 8 Arquitectura Orientada a Servicios

2.2. Arquitectura de software de IoT

Para el desarrollo de aplicaciones IoT existen dos arquitecturas de uso general como las mostradas en [12]:

La arquitectura de tres capas es la más sencilla (Figura 9 a) y consiste en las capas denominadas percepción, red y aplicación. La capa de percepción contiene los dispositivos que recopilan datos del entorno físico mediante el uso de sensores. Se encarga de recolectar los datos y enviarlos a la capa de red. La capa de red es la encargada de conectar la capa de percepción con el entorno físico. Debe de contar con tecnología para conectar los dispositivos de forma alámbrica o inalámbrica mediante canales seguros de comunicación. Los datos recibidos por la capa de aplicación serán analizados y procesados para proporcionar servicios y así poder tomar decisiones.

Por otro lado, la arquitectura de cinco capas (Figura 9 b) es una adecuación de la arquitectura de tres capas, a la que se le adicionan las capas denominadas Puerta de Enlace (del inglés *Gateway*) y Capa Intermedia (del inglés *Middleware*). La capa de Gateway es la responsable del manejo de dispositivos IoT y de efectuar el intercambio de mensajes entre los dispositivos y los diversos subsistemas. Mientras que la capa del *Middleware* proporciona un vínculo flexible para la comunicación entre el conjunto de los dispositivos y las aplicaciones.

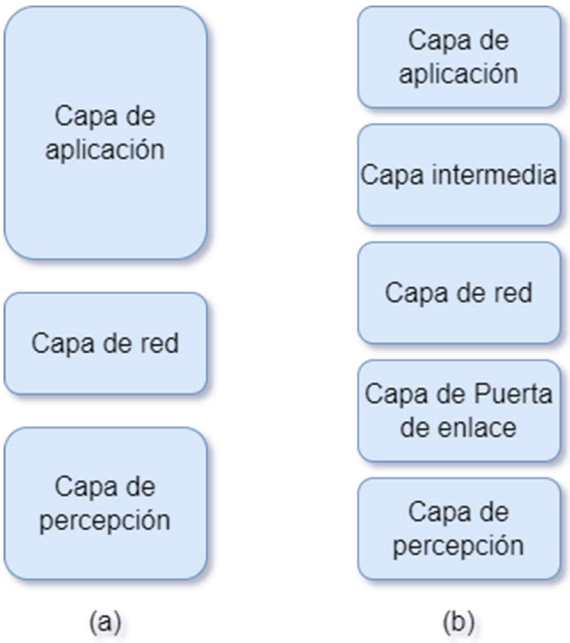

Figura 9 Arquitectura de 3 capas (a) y 5 capas (b)

2.3. Arquitectura de software de IoT orientada a servicios

"La arquitectura orientada a servicios (SOA) ofrece un conjunto de patrones y pautas para crear servicios de manera flexible y alineados con el negocio. Permite separar la especificación, la implementación y los enlaces, con la finalidad de prevenir amenazas y aprovechar oportunidades comerciales*"* [4].

La arquitectura orientada a servicios surge como una promesa para cerrar la brecha entre los dispositivos industriales y las aplicaciones empresariales. El funcionamiento general consiste en ofrecer interfaces de comunicación entre distintas aplicaciones. Las interfaces o servicios son las encargadas de definir métodos para obtener un resultado de los datos ingresados. Cada servicio es independiente y pueden ser interconectados con otros para mejorar diversos aspectos del servicio. Esta arquitectura tiene como objetivo el tener flexibilidad, desarrollo distribuido y administración.

De acuerdo con [13] en este patrón se consideran siete capas:

1. Capa de sistemas operacionales.- En esta capa se integran los sistemas existentes utilizando técnicas de integración especializadas de SOA.
2. Capa de componentes empresariales.- Es la responsable de agregar la funcionalidad y mantener la calidad de los servicios que se ofertan a los clientes.
3. Capa de servicios.- Son los servicios que el negocio elige para estar en esta capa, proporciona los mecanismos para tomar componentes a escala empresarial, componentes específicos de la unidad de negocio y componentes específicos del proyecto.
4. Capa de composición del proceso empresarial.- En esta capa los servicios se agrupan en un flujo a través de la orquestación o la coreografía y actúan juntos como una sola aplicación.
5. Capa de presentación.- Proporciona el acceso punto a punto a los servicios.
6. Capa de integración.- Permite la integración de servicios a través de la introducción de un conjunto confiable de capacidades como el enrutamiento inteligente, la mediación de protocolos y otros mecanismos de transformación.
7. Capa de calidad del servicio (QoS).- Proporciona las capacidades requeridas para monitorear, administrar y mantener la QoS, como la seguridad, rendimiento y disponibilidad. Todo esto realizado en segundo plano a través de mecanismos y

herramientas de detección y respuesta que monitorean el estado de las aplicaciones SOA.

2.4. Requisitos a considerar en la arquitectura

Para este proyecto se considera primordial el poder integrar, modificar o deshabilitar servicios sin alterar el funcionamiento general del sistema. Para ello se deben de buscar atributos de calidad relacionados con la modificabilidad e interoperabilidad en dichos servicios.

La modificabilidad del sistema se puede satisfacer mediante la segmentación de los componentes en módulos que sean independientes de otros, así, al modificar o reemplazar un módulo éste afecte a la menor cantidad posible y a la vez permitir que el mantenimiento de los servicios sea más sencillo.

Al igual que la modificabilidad, la interoperabilidad en el sistema se satisface con una estandarización en los medios de comunicación entre los módulos, al utilizar protocolos de comunicación estandarizados se facilita la interconexión de componentes y da la posibilidad a una mayor interacción con otros sistemas que puedan necesitar la interacción.

Tomando en cuenta ambos atributos de calidad se debe buscar un patrón arquitectónico que nos permita satisfacer las necesidades del proyecto.

2.5. Componentes y servicios

Buscando satisfacer los atributos de calidad mencionados anteriormente, se plantea el desarrollo de componentes y servicios en conjunto con la utilización de los componentes de las plataformas FIWARE y DeviceHive para construir un sistema que permita integrar, modificar o deshabilitar los servicios que se proporcionan para la localización de personas en espacios cerrados.

Como componente principal para la visualización se desarrollará una aplicación web en PHP a la cual se añadirán los servicios en formato modular de visualización y alertas visuales para el usuario que indicarán con distintos colores la proximidad a la zona externa del mapa donde no sea monitorizado por la aplicación. Para el caso de los componentes de las plataformas se optó por mantener la mínima cantidad de componentes de cada plataforma

para un funcionamiento similar, lo cual consiste en un bróker para la comunicación, un almacenamiento de datos, y una interfaz de comunicación con los dispositivos.

Se plantea el uso de una arquitectura compuesta que tenga de base el modelo de capas lo cual permitirá el uso de módulos y submódulos para permitir la modificación a los servicios que darán la funcionalidad al sistema, y a su vez permitir la comunicación con las plataformas de FIWARE y DeviceHive que a su vez se comunicaran con los dispositivos móviles y el servicio de posicionamiento correspondiente. Los detalles de diseño e implementación de la arquitectura se describirán más adelante en los capítulos 4 y 5 respectivamente.

2.6. FIWARE

La plataforma de FIWARE [14], [15]ofrece un amplio catálogo de componentes de código abierto que se pueden ensamblar junto con otros componentes de terceros para construir plataformas que permitan el desarrollo de soluciones inteligentes de una forma más rápida y sencilla. El componente fundamental que debe tener cada plataforma o solución funciona como un habilitador genérico de agente de contexto FIWARE, que proporciona una función fundamental requerida en cualquier solución inteligente.

El principal componente y de uso obligatorio es el denominado Orion Context Broker (OCB), el cual permite la publicación de información de contexto por entidades productoras, de manera que la información pública se encuentra disponible para otras entidades consumidoras que están interesadas en procesar la información.

FIWARE NGSI es la API del OCB, ésta es utilizada para la integración de componentes de la plataforma y por aplicaciones para actualizar o consumir información de contexto. En el FIWARE Context Broker se encuentra disponible un amplio conjunto de habilitadores genéricos de código abierto que se ocupan de lo siguiente:

- Proporcionar una interfaz con el internet de las cosas, robots y sistemas de terceros, para capturar actualizaciones sobre información de contexto y realizar las acciones necesarias.

- Realizar la gestión, publicación y monetización de datos de contexto / API, brindando soporte al control de uso y a la oportunidad de publicar y monetizar parte de los datos de contexto admirativos.
- Procesamiento, análisis y visualización de información de contexto, implementando el comportamiento inteligente esperado de las aplicaciones y ayudando a los usuarios finales a tomar decisiones inteligentes.

La plataforma no cuenta con la restricción de usar todo o nada. El usuario jamás es obligado a usar todos los componentes o habilitadores genéricos que la plataforma proporciona, se permite combinar los componentes con otras plataformas de terceros para crear una plataforma híbrida que se adapte a las necesidades de cada proyecto.

Al utilizar la tecnología FIWARE Context Broker para la gestión de información de contexto, la plataforma desarrollada se puede etiquetar como "*Powered by FIWARE*" y así poder incluir el desarrollo en el FIWARE Marketplace para que otras personas puedan utilizar o contribuir a la plataforma.

2.7. DeviceHive

La plataforma DeviceHive [16], [17]es una plataforma de código abierto que proporciona instrumentos para la comunicación y gestión de dispositivos inteligentes. Es una plataforma basada en microservicios que cuenta con una API de administración, que utiliza diversos protocolos, que permite configurar y monitorear los dispositivos conectados. Los componentes principales que ofrece se describen a continuación:

- Kafka (Bus de mensajería).- Se encarga de la comunicación entre los servicios y también maneja el balance de carga entre ellos.
- DeviceHive Frontend.- Es un servicio que proporciona conectividad mediante protocolos RESTful y *Websocket* APIs, se encarga de mandar peticiones al servicio de Backend y a la transmisión de respuestas de forma asíncrona.
- Hazelcast IMDG.- Proporciona un almacenamiento de datos que requieren ser accedidos con velocidad, toda notificación es guardada en memoria cache y después es movida al almacenamiento general después de un tiempo específico.

- PostgreSQL.- Proporciona almacenamiento para toda la información persistente en el sistema.
- DeviceHive Backend.- Es el servicio responsable de almacenar datos, administrar subscripciones y recuperar información de otros servicios.
- Servicio de Autenticación.- Permite validar el acceso a la plataforma mediante JWT tokens, los cuales contienen los privilegios del usuario, los dispositivos disponibles y los tipos de redes o dispositivos. Proporciona una API para la generación, validación y actualización de tokens.

En cuestión de comunicación la plataforma proporciona protocolos REST, APIs web y MQTT, toda información es realizada mediante mensajes JSON. Casi en su totalidad de dispositivos que utilicen esos protocolos de comunicación pueden interactuar directamente con DeviceHive, los dispositivos que puedan utilizar Python, Node.js o Java pueden conectarse de manera sencilla al instalar la librería de cliente en la plataforma.

Al ser una plataforma de código abierto y que está basada en una arquitectura de microservicios, se permite la modificación de componentes e intercambiarlos con cualquier otro que sea conveniente al usuario. Otra forma de modificación que ofrece la plataforma es el uso de *plug-in* para la suscripción de notificaciones del sistema, para así conectar la información con otra plataforma al mismo tiempo.

2.8. Comparación de las plataformas FIWARE y DeviceHive

Para el trabajo propuesto se seleccionaron las plataformas FIWARE y DeviceHive como servicios para la capa de IoT de la arquitectura, esto debido a las similitudes que presentan ambas plataformas y para fines prácticos de la investigación ambas son de código abierto para utilizar según sea conveniente. A continuación, se comparan las características de ambas plataformas en los aspectos de confiabilidad, escalabilidad, heterogeneidad y usabilidad.

En el trabajo [18] los autores realizaron una comparación entre múltiples plataformas de IoT comerciales y de software libre. De estos datos se obtiene la siguiente información correspondiente a las plataformas FIWARE y DeviceHive.

2.8.1 Confiabilidad

En el aspecto de confiabilidad la replicación es fundamental al momento de trabajar con datos en tiempo real, por lo cual es importante replicar la información de los datos en movimiento, así como los almacenados, una solución a este problema es distribuir servidores para replicar los datos y almacenarlos y a la vez consiguiendo mejores tiempos de acceso para los usuarios.

Sabiendo que existe esta clase de solución, la plataforma FIWARE sugiere un despliegue de múltiples nodos computacionales, que además de replicar los datos añade una función de clúster para mejorar el rendimiento en sus servicios, también ofrece la posibilidad de añadir múltiples nodos de almacenamiento, aunque estos no cuenten con mecanismos de replicación de datos en automático.

En cuestión de disponibilidad de infraestructura y servicios, ambas plataformas pueden ser desplegadas en cualquier servidor siempre que cumpla las características mínimas para cada plataforma, por lo cual es responsabilidad del administrador manejar las estrategias para asegurar un funcionamiento óptimo en el sistema. Considerando esto, FIWARE sugiere una arquitectura de despliegue que incluye tres controladores que permiten mantener una alta disponibilidad la mayor parte del tiempo.

Como se sabe, la seguridad es una cuestión importante al trabajar con sistemas distribuidos y más aún si estos utilizan redes públicas para mantener la comunicación. La plataforma DeviceHive utiliza JSON *Web Tokens* (JWT) como método de autentificación, que a su vez incluye información de autorización. Por otro lado, FIWARE optó por utilizar OAuth2 en conjunto con *Keyrock identity manager* como mecanismos para el manejo de información de autenticación. Si bien utilizan distintas tecnologías, ambas plataformas implementan la autorización basada en roles para gestionar permisos.

Al igual que el punto anterior, la encriptación de los mensajes y los datos forma parte esencial de la seguridad. DeviceHive ofrece encriptación en la comunicación por TLS, a diferencia de FIWARE que no proporciona directamente mecanismos de encriptación.

Un software puede mantenerse confiable a medida que se siga desarrollando, por lo cual es importante destacar que tan frecuente se realizan actualizaciones para corregir errores o agregar funcionalidades a las plataformas. Por un lado, DeviceHive muestra un mapa de desarrollo, el cual realizaba actualizaciones frecuentes, esto lo siguió realizando hasta el 2018. En cuanto a FIWARE, actualmente se encuentra distribuido en múltiples repositorios,

donde gracias a la comunidad de la misma plataforma, se siguen actualizando algunos componentes para mejorarlos cada día, cabe destacar que también existen algunos componentes que no se modifican desde hace años.

2.8.2. Escalabilidad

La escalabilidad es la habilidad del software de crecer y manejar el incremento de la demanda. El hosting de la plataforma impactará de manera directa la escalabilidad de la plataforma y otros aspectos importantes, un ejemplo de esto es, al desplegar los servicios en la nube se cuenta con un mayor acceso a recursos a comparación de un servidor local, en cambio una implementación local o de nube dedicada cuenta con mayor seguridad debido a la separación física o lógica que esta pueda tener.

Ambas plataformas pueden ser desplegadas tanto de forma local, como en la nube, todo depende de las necesidades del cliente y de las capacidades del administrador para lograr una gestión correcta de recursos y configuraciones. Adicionalmente, FIWARE ofrece *FogFlow*, un orquestador para *cloud-edge* para lograr reducir el tiempo entre las acciones y reducción de costos y ancho de banda.

2.8.3. Heterogeneidad

Para lograr mantener la heterogeneidad en las plataformas IoT y en sus redes, se deben implementar estándares para la comunicación e implementación. Las plataformas de software libre no cuentan con restricciones de dispositivos para su implementación, para DeviceHive es necesario realizar un despliegue en un sistema que utilice contenedores.

Proporcionar librerías o kits de desarrollo de software (SDK, por sus siglas en inglés) puede simplificar el proceso de desarrollo de software con esas tecnologías, y a su vez ayuda a mantener un estándar en el código. En el caso de DeviceHive, la plataforma ofrece una amplia variedad de SDKs para los lenguajes Java, JavaScript, C++, C#, Go, Python, Android y iOS.

Entre más interfaces de comunicación pueda proveer una plataforma incrementa significativamente su nivel de heterogeneidad, tanto DeviceHive como FIWARE utilizan los protocolos más usados como el HTTP, MQTT, así como WebSockets y en el caso de

FIWARE añade un protocolo NGSI. En cuestión de tecnologías de conexión con los dispositivos, ambas manejan *Bluetooth* y *ZigBee*.

En términos de flexibilidad para mantener la heterogeneidad en los sistemas IoT, ambas plataformas requieren el uso de un núcleo de procesamiento y opcionalmente añaden funcionalidades con *plug-ins* para DeviceHive o habilitadores genéricos en FIWARE.

2.8.4. Usabilidad

Una plataforma por más que cuente con todos los atributos anteriormente mencionados aún es necesario que ofrezca una buena usabilidad para el usuario. Incluir documentación, tutoriales o ejemplos permite al usuario tener un panorama más amplio para entender las capacidades de cada plataforma.

La documentación que presenta DeviceHive es fácil de entender, pero no se profundiza en los detalles técnicos, cuenta con un laboratorio virtual para hacer pruebas desde la misma página web o aplicaciones externas. Ofrece plantillas para desarrollo en múltiples lenguajes y el soporte al usuario se realiza mediante preguntas y respuestas en sus sitios web, donde la comunidad puede participar libremente.

En términos de documentación FIWARE proporciona una documentación más extensa con una buena cantidad de ejemplos por parte de la plataforma, aunque algunos usuarios experimentados afirman que aún le falta para cumplir con una buena documentación. El soporte a usuarios se realiza de la misma manera mediante la realización de preguntas y respuestas en sitios web por parte de la comunidad y expertos en FIWARE.

En resumen, ambas plataformas cuentan con características muy similares que ayudarán al desarrollo de este proyecto, si bien tienen algunas diferencias, estas características no son utilizadas para el proyecto para mantener el mismo nivel de tecnología en ambas plataformas y así mantener los resultados de las evaluaciones lo más equivalente posible.

2.9. Servicios Web

En [19] se describen los servicios web como tecnologías basadas en XML para mensajería, descripción de servicios, descubrimiento y características extendidas, que proporcionan:

- Estándares abiertos y generalizados para las descripciones de la interfaz informática distribuida y el intercambio de documentos a través de mensajes.
- Independencia de la tecnología de ejecución y de las plataformas de aplicación.
- Extensibilidad para las calidades empresariales del servicio, como seguridad, confiabilidad y transacciones.
- Soporte para aplicaciones compuestas tales como flujo de procesos de negocios, acceso multicanal e integración rápida.

De esto se entiende que los servicios web otorgan a la arquitectura orientada a servicios una mayor facilidad de comunicación entrante o saliente para que esta se adapte a diversos sistemas y plataformas existentes, creando así sistemas con una mayor compatibilidad y reusabilidad.

2.10. Interfaz de programación de aplicaciones

De acuerdo con [20], una Interfaz de programación de aplicaciones (API, del inglés *Application Programming Interface*) "es un intermediario de software que permite que dos aplicaciones se comuniquen entre ellas". Una API puede describirse como una interfaz de conexión genérica para aplicaciones. las APIs constan de las siguientes características:

- Uso de estándares (HTTP y REST).
- Diseñadas para consumo de datos.
- Estandarizadas.

2.11. Edificios inteligentes

De acuerdo con [21], se considera como edificio inteligente a aquel que cuenta con un diseño que maximiza la funcionalidad de este. Permite, además, incorporar y modificar los elementos necesarios para el desarrollo de la actividad cotidiana, con la finalidad de garantizar una mayor productividad estimulada por un ambiente de máximo confort. La

automatización de las instalaciones, así como la integración de los servicios, son algunos de los objetivos con los que generalmente cuenta un edificio inteligente.

Le y Al Dakheel [22], [23]explican que los edificios inteligentes deben de contar con cinco características principales:

- Automatización.- la habilidad de permitir que múltiples dispositivos realicen tareas o funciones de manera autónoma.
- Multifuncionalidad.- la habilidad de permitir la ejecución de más de una función dentro del edificio.
- Adaptabilidad.- la habilidad de aprender, predecir y satisfacer las necesidades de los usuarios.
- Interactividad.- la habilidad de permitir interacciones entre los usuarios.
- Eficiencia.- la habilidad de proporcionar energía de forma eficiente para reducir costos y tiempo.

De lo anterior se concluye que la funcionalidad de monitoreo de personas en un edificio abre un abanico de posibilidades para la implementación de sistemas de salud, productividad, consumo energético, automatización, entre otras.

2.12. Geolocalización

La geolocalización, también conocida como localización o posicionamiento, “es la capacidad de identificar la posición geográfica de un objeto, persona o dispositivo móvil en el espacio real”[24]. Este término está estrechamente relacionado con el uso de sistemas de posicionamiento, pero puede distinguirse de estos por un mayor énfasis en la determinación de la posición significativa (por ejemplo, la dirección de una calle) y no solo por un conjunto de coordenadas geográficas [25].

Estrictamente, la geolocalización es la posición de un objeto en un espacio, lo importante es destacar el grado de precisión de esta medición. Para los sistemas de navegación y localización, la precisión es clave para el correcto funcionamiento.

2.12.1. Localización en espacios cerrados

La localización en espacios cerrados es la capacidad de proporcionar la ubicación de un objeto, persona o dispositivo móvil dentro de un edificio. Puede utilizar las mediciones de diversos sensores de un teléfono inteligente, como el WiFi, Bluetooth, magnetómetro, giroscopio, o acelerómetro, para obtener la localización. El teléfono inteligente detecta las variaciones de las señales de los sensores y ejecuta una serie de algoritmos para estimar la posición y el comportamiento del usuario [26].

La localización en espacios cerrados proporciona una mayor precisión gracias a la combinación de múltiples fuentes de información distribuidas en el entorno y los algoritmos responsables para su estimación. La posibilidad de usar diversas fuentes de información permite la incorporación de sistemas más especializados con dichas tecnologías, brindando así una mayor precisión según sea necesario.

2.13. Tecnologías emisoras de señales inalámbricas

2.13.1. WiFi

De acuerdo con [27] la tecnología WiFi se basa en el estándar de comunicación inalámbrica 802.11 del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE, del inglés *Institute of Electrical and Electronics Engineers*). “La tecnología WiFi evoluciona continuamente ya que cada generación ofrece velocidades más rápidas, menor latencia y mejores experiencias y es la tecnología de comunicación inalámbrica más usada comúnmente, así como el principal medio para el tráfico global de internet”.

Al ser una tecnología en constante demanda por todo tipo de usuarios, es una herramienta para usar como fuente de información para la estimación de objetos que puedan captar los niveles de intensidad de las redes WiFi.

2.13.2. Bluetooth de baja energía

El bluetooth de baja energía (BLE, del inglés *Bluetooth Low Energy*) está diseñado para funcionar con un muy bajo consumo de energía. Permite el funcionamiento confiable en la frecuencia de 2.4 gigahertz, aprovechando un enfoque de alto espectro de salto de frecuencia que transmite datos a través de 40 canales disponibles.

Esta tecnología admite múltiples topologías de red, incluida una opción de punto a punto utilizada para transferencia de datos, una opción de difusión utilizada para servicios de ubicación y una opción de malla para crear redes de dispositivos a gran escala [28].

Capítulo 3

Revisión de la literatura

3. Revisión de la literatura

En este capítulo se describen los trabajos relacionados que fueron identificados para la presente investigación. Los trabajos seleccionados abarcan temas desde tecnologías, herramientas, plataformas y arquitecturas para el internet de las cosas y su aplicación en la localización en interiores.

3.1. Antecedentes en CENIDET

Para la localización de personas y objetos en espacio abiertos se utiliza la tecnología GPS que se encuentra en los teléfonos inteligentes; sin embargo, dicha tecnología no funciona con la misma precisión en espacios cerrados, esto se debe a la pérdida de señal que sufre al tener que atravesar objetos sólidos de las edificaciones [2].

En la revisión de los trabajos previos realizados en la institución, encontramos dos artículos [29], [30], en los cuales los autores presentan dos aplicaciones que basan el cálculo de la ubicación utilizando emisores BLE (del inglés *Bluetooth Low Energy*), las cuales presentan la ventaja de dar seguimiento de manera continua y así poder efectuar diversos estudios de movilidad de personas en espacios cerrados. Asimismo, se encontraron las tesis concluidas en los años 2009, 2012 y 2015 respectivamente [31]–[33], relacionadas con la navegación en interiores utilizando tecnologías como los códigos QR y tarjetas RFID, en el que presentan algunas desventajas para el propósito de esta tesis, debido a que estas tecnologías no permiten el monitoreo de la ubicación de manera continua.

Cervantes [30], propone un sistema de localización en espacios cerrados basado en el algoritmo de trilateración y tecnología BLE (del inglés *Bluetooth Low Energy*). En espacios cerrados, la utilización de la tecnología BLE tiene la ventaja de menor consumo de energía y mayor precisión con respecto a los sistemas de posicionamiento GPS. Para el cálculo de ubicación se obtiene la intensidad de la señal recibida que emiten los dispositivos BLE, para determinar, por medio del algoritmo de trilateración, la ubicación del dispositivo dentro de un recinto cerrado. Los resultados de la precisión de la ubicación en un área cerrada libre de obstáculos son aceptables, sin embargo, si la señal es obstaculizada por muros sólidos, ésta se degrada obteniendo un margen de incertidumbre mayor.

En el trabajo [29], Martínez presenta una propuesta para la navegación en espacios cerrados que utiliza la tecnología BLE y teléfonos inteligentes. Los autores desarrollaron una aplicación móvil que obtiene información del contexto que la rodea y genera la ruta óptima

para llegar a un destino a partir de una ubicación inicial. La investigación utilizó la intensidad de las señales emitidas por los dispositivos BLE para determinar la ubicación del usuario. La generación de la ruta óptima fue realizada con el algoritmo de Dijkstra. La ruta óptima de la persona para llegar desde un nodo origen a un nodo destino se muestra por medio de mensajes de texto, el sistema no cuenta con interfaz gráfica que posicione al usuario en un plano de dos dimensiones. En el trabajo no se menciona el margen de incertidumbre en el cálculo de la ubicación.

En la tesis [31] Yris modela, desarrolla, prueba y muestra una interfaz de programación de aplicaciones (API, del inglés *Application Programing Interface*) capaz de reconocer patrones de señales electromagnéticas de las tecnologías de comunicación más comunes (WiFi, Bluetooth y RFID), para poder obtener la localización de un objeto en espacios cerrados. La API lleva por nombre CHMAN y está compuesta por cuatro capas (capa de conexión física, capa de traducción, capa de filtrado y capa de procesamiento de datos). Esta investigación proporciona la ubicación del usuario con un margen de incertidumbre en la posición de 2 a 9 metros.

En la tesis [32] Ramírez presenta una aplicación para la navegación en interiores combinada con realidad aumentada, interactuando con puntos de interés (lugares, personas, eventos, etc.). En esta investigación se desarrolló un sistema de navegación capaz de guiar a un usuario dentro y fuera de una organización mediante reconocimiento de visión artificial. Muestra diferentes objetos con técnicas de realidad aumentada y decodificación de código QR. Posteriormente, los autores realizaron una implementación del sistema propuesto mediante una aplicación para el sistema operativo Android, el cual fue probado dentro de las instalaciones del plantel, utilizando los sensores del dispositivo (cámara para la visión artificial y el GPS para la determinación del lugar dentro del edificio). En este trabajo no se menciona el margen de incertidumbre en el cómputo de la ubicación, pero sin embargo, es interesante como la realidad aumentada puede ayudar al posicionamiento.

En la tesis [33] el trabajo de Arjona consistió en un sistema de trazado de ruta automático dentro de un edificio multinivel y en áreas del tipo campus como universidades, conjuntos de edificios empresariales o supermercados, con base en la posición del usuario y una tarea determinada. El sistema muestra, mediante un mapa, la ubicación del usuario en tiempo real y la ubicación final donde se lleva a cabo la tarea en cuestión. En esta

investigación el desarrollo se llevó a cabo en teléfonos inteligentes con sistema operativo Android, utilizando las tecnologías de localización RFID y códigos QR. En este proyecto no se consideró importante mencionar el margen de incertidumbre para el cálculo de posición.

En la tesis en curso[34], Infante propone desarrollar una aplicación capaz de proporcionar a los usuarios una localización con un menor margen de incertidumbre en los espacios cerrados, utilizando algoritmos específicos como fingerprinting y trilateración, dependiendo de las entradas de señales inalámbricas. También se propone una guía para la colocación estratégica de los dispositivos emisores de señales.

La revisión de los trabajos realizados en el CENIDET ha demostrado la importancia de desarrollar sistemas de localización en espacios cerrados que ofrezcan una mayor precisión en comparación con la tecnología GPS. Las tecnologías WiFi y BLE se han presentado como alternativas competentes en conjunto de algoritmos de ubicación selectos como trilateración y fingerprinting. En general, los trabajos previos han proporcionado una base sólida para el desarrollo de una arquitectura para la localización en espacios cerrados, pero aún es deficiente la implementación ágil de estos servicios de forma generalizada.

3.2. Revisión de la literatura

En esta Sección, se presenta el análisis de los artículos de arquitecturas de software, arquitecturas orientadas al internet de las cosas, tecnologías inalámbricas y su utilización para el cálculo de la posición en espacios cerrados.

En esta sección se aplicaron los criterios para describir cada investigación relacionada con el tema de este documento, los criterios que se utilizaron fueron:

- Descripción general de la investigación. Se describen los aspectos generales de la investigación, así como sus aportaciones principales.
- Tecnologías utilizadas. Se describe el tipo de tecnología u objetos de estudio que se utilizaron en la investigación.
- Resultados. Se describen los resultados de los experimentos, y las conclusiones de los autores.
- Conclusiones. Se describen las conclusiones, ventajas y desventajas que se identificaron en el trabajo, y se describe cómo puede ser de utilidad para este documento.

3.2.1. Rendimiento y desafíos de la arquitectura orientada a servicios para redes de sensores inalámbricos

Descripción general de la investigación

En esta investigación [35] los autores presentan dos arquitecturas como solución a los desafíos que se enfrentan para el desarrollo de aplicaciones IoT. Tanto la Arquitectura Orientada a Servicios (SOA, del inglés *Service Oriented Architecture*) y el Middleware Orientado a Servicios (SOM, del inglés *Service Oriented Middleware*) proporcionan soluciones para los desafíos de las Redes de Sensores Inalámbricos (WSN, del inglés *Wireless Sensor Network*).

Tecnologías utilizadas

En esta investigación los autores realizaron un análisis comparativo de las diversas arquitecturas basadas en SOA y SOM, donde se destacan las ventajas y desventajas, así como también los requerimientos para su implementación y los beneficios que proporciona. Los puntos más importantes en cada categoría son:

- SOM. La mayoría de las arquitecturas SOM para las WSN se basan en servicios heterogéneos. Estos servicios afectan el tiempo de respuesta y la eficiencia de la red. Esta clase de arquitectura trata con cantidades masivas de mensajes y eventos de varios servicios que comparten esos mensajes y eventos entre los componentes del sistema.
- SOA. Los desafíos de seguridad y el rendimiento de la agregación de datos no son compatibles con la mayoría de los enfoques. Los autores concluyen que ninguno de los enfoques revisados cumple todos los requisitos a nivel global, por lo cual se debe considerar el desafío que se desea superar al momento del diseño de la arquitectura.

Resultados

La principal contribución de este documento es el diseño, implementación y validación de la arquitectura SOM para diversas aplicaciones y entornos basados en tecnologías WSN. Por lo

tanto, estas arquitecturas están diseñadas para considerar las complejidades relacionadas con los recursos de las redes de sensores.

Conclusiones

Se debe considerar un mecanismo para el descubrimiento de servicios, el cual debe estar disponible para asegurar la continuidad del servicio, descubriendo así cualquier falla y reemplazándola con el mejor servicio disponible durante el tiempo de ejecución. Se debe considerar al momento de realizar el diseño de la arquitectura los principales desafíos a superar.

3.2.2. Un enfoque cuantitativo para analizar la modificabilidad en el diseño arquitectónico de software de sistemas de aplicaciones ágiles

Descripción general de la investigación

En esta investigación [36] los autores presentan una arquitectura compuesta que busca dar solución a las problemáticas actuales referentes al desarrollo ágil de proyectos de software y a la modificabilidad de estos. Mediante el uso de distintos patrones arquitectónicos, los autores combinan los componentes de las arquitecturas como el Modelo Vista Controlador, Tuberías y Filtros y la Arquitectura Reflejante, creando así una arquitectura compuesta genérica enfocada en aplicaciones en tiempo real. Así mismo, los autores proponen una métrica para evaluar la modificabilidad dentro de una arquitectura en términos cuantitativos.

Tecnologías utilizadas

En esta investigación los autores diseñaron una arquitectura compuesta para aplicaciones en tiempo real, en la cual evaluaron el atributo de modificabilidad con la métrica que diseñaron, con base en técnicas del Método de análisis de compensación arquitectónica (ATAM, por sus siglas en inglés), el Método de análisis de arquitectura de software (SAAM, por sus siglas en inglés), y el Análisis de modificabilidad a nivel de arquitectura (ALMA, por sus siglas en inglés).

El patrón de la arquitectura compuesta utiliza los patrones de otras arquitecturas como base para generar una arquitectura genérica que satisface las necesidades del proyecto. Los componentes individuales utilizados se describen a continuación:

- **Modelo interactivo**.- Utiliza el patrón de modelo vista controlador, el cual proporciona una interfaz para permitir que el usuario ingrese los parámetros iniciales de ejecución.
- **Procesamiento de datos en tubería**.- Utiliza el patrón reflejante, el cual permite realizar cambios dinámicos a la estructura y al comportamiento del sistema.

Para determinar que una arquitectura de software cumple con los atributos de calidad es necesario realizar una evaluación de estos. Los factores como el acoplamiento, la cohesión y la complejidad se utilizan para medir la modificabilidad de una arquitectura. Los autores calculan la modificabilidad con respecto a los valores de acoplamiento y complejidad. A continuación, se describen las ecuaciones y los parámetros utilizados para el cálculo de la modificabilidad:

N.- Número de filtros en el sistema.

p.- Número de enlaces directos entre filtros.

q.- Número de enlaces indirectos entre filtros.

k.- Constante entera que indica un conjunto de filtros que tienen una relación

El acoplamiento es definido como la relación del número total de enlaces directos entre el total de enlaces posibles (directos o indirectos):

$$(1) Acoplamiento = p/(p+q)$$

La complejidad ciclomática se define como:

$$(2) Complejidad\ Ciclomática\ (CC) = E - N + 2$$

Donde E es el número de enlaces entre filtros y N es el número de nodos o filtros

$$(3) p + q \geq k$$

Sustituyendo la ecuación 3 en la ecuación 1 se obtiene:

$$(4) Acoplamiento \leq p/k$$

Para calcular la complejidad ciclomática, el número de aristas(E) corresponde al número de enlaces directos entre los filtros, por lo tanto:

$$(5) CC = p - N + 2$$

Sustituyendo la ecuación 4 en la ecuación 2 se obtiene:

$$(6) Acoplamiento * k - N + 2 \leq 10$$

$$Acoplamiento = (8+N)/k$$

$$(7) Modificabilidad \propto \left(\frac{1}{Acoplamiento}\right)$$

$$Modificabilidad = M * \left(\frac{1}{Acoplamiento}\right)$$

Donde *M* es una constante proporcional que se asumió un valor de 1 en este estudio realizado por los autores. Como resultado los autores obtienen una tabla para calcular la modificabilidad.

Tabla 1. Valores de Modificabilidad en una arquitectura

Factor de acoplamiento	Umbral de modificabilidad	Evaluación arquitectónica
[0]	Muy alto	Excelente
(0-0.5)	>2	Bueno
[0.5]	2	Altamente aceptable
(0.5-1)	1<modificabilidad<2	Aceptable
[1]	1	No aceptable

Resultados

El algoritmo propuesto por los autores fue implementado satisfactoriamente para un sistema de procesamiento de documentos JAVA en un sistema operativo Ubuntu 12. De acuerdo con la tabla y los resultados mostrados, los autores obtuvieron un valor de umbral de modificabilidad de 1.5, lo cual nos indica que el diseño de la arquitectura que realizaron fue aceptable en cuestión de modificabilidad. Se establece la relación entre la modificabilidad y el número de conexiones directas entre filtros. Los autores concluyen que, con el incremento de las conexiones en paralelo, el factor de acoplamiento entre filtros es muy bajo, lo cual incrementa la modificabilidad de la arquitectura.

Conclusiones

Este trabajo es muy importante al momento de la evaluación de la arquitectura que se diseñará en este trabajo. Es cuestión de adaptar las fórmulas para que se adapte a las necesidades de la arquitectura compuesta a diseñar. Ya que se plantea utilizar una arquitectura compuesta para solventar la problemática actual y que pueda ser utilizada con diversos componentes que existen actualmente en el mercado. Este trabajo evalúa matemáticamente la modificabilidad de la arquitectura, lo cual permite dar una prueba inequívoca de que se cumple con el propósito principal de la arquitectura, cuyo objetivo principal es la modificabilidad en los servicios que proporciona al usuario.

3.2.3. Investigación sobre la arquitectura general del middleware de Internet de las cosas para parques industriales inteligentes

Descripción general de la investigación

En esta investigación [37] los autores presentan el uso de una tecnología de middleware basada en una arquitectura orientada a servicios, para la generación de un modelo de Middleware orientado a servicios. El trabajo muestra un caso de estudio, detallando las funciones que deben de ser implementadas en las capas importantes del proyecto, así como su implementación. Se muestra una clasificación de los Middleware, así como su función principal y una descripción de las características de una arquitectura orientada a servicios.

Tecnologías utilizadas

En el trabajo los autores utilizaron la tecnología denominada Middleware para el internet de las cosas, la cual se maneja con base en eventos de sistema, los datos obtenidos de diversas fuentes son transformados en un formato estándar XML.

- **Control de dispositivos**.- Proporciona una serie de interfaces para la comunicación de datos con los dispositivos y la capa de programación de servicios. El módulo de filtrado de datos que se encuentra en esta capa es el responsable de la recolección de datos de los dispositivos, permitiendo modificar la periodicidad de la actualización de los datos.
- **Programación de servicios**.- Se divide en módulo de entrega de eventos y módulo de registros del sistema. El módulo de entrega de eventos se divide en dos partes, una encargada de escuchar los eventos que surgen de la capa de control de dispositivo, y la segunda parte se encarga de escuchar las llamadas desde la capa de aplicaciones de negocio para que se realice la tarea seleccionada. El módulo de registro del sistema permite mantener un registro de las acciones en el sistema, lo cual ayuda en la solución de problemas y también para observar el comportamiento general del sistema.
- **Aplicaciones de negocio**.- Es el responsable de interactuar directamente con la aplicación superior o la base de datos. Debido a que la capa superior puede tener

muchas aplicaciones diferentes, se debe definir una función API de publicación externa estándar.

Resultados

La arquitectura propuesta por los autores satisface los requisitos de múltiples accesos de dispositivos Zigbee IoT, así como también múltiples aplicaciones para el monitoreo de una fábrica.

Conclusiones

Se puede considerar para el uso en el proyecto la arquitectura middleware IoT que consiste en tres capas: La capa de controlador de dispositivo, la cual debe de utilizar protocolos de transmisión y formato en los datos dependiendo del dispositivo, para así buscar una interfaz de datos unificada. La capa de programación de servicios debe proporcionar un sistema de registro y administración de toda la plataforma de middleware. Por último, la capa empresarial de aplicaciones debe brindar varios métodos de interacción de datos con el sistema de aplicación y a su vez diseñar una interfaz unificada en la capa empresarial de aplicación.

3.2.4. Soluciones de Amazon, Google y Microsoft para IoT: arquitecturas y una comparación de rendimiento

Descripción general de la investigación

En esta investigación [38] se comparan las tres principales plataformas en la nube (Amazon Web Services, Google Cloud Platform y Microsoft Azure) con respecto a los servicios disponibles para IoT. Se describen las arquitecturas y se analizan los puntos clave de cada una.

Tecnologías utilizadas

En el trabajo se analizan los puntos clave de cada arquitectura, así como también se analiza el rendimiento en las tres plataformas comparándolas con un servicio que tienen en común (Middleware MQTT).

- AWS IoT Core.- Cuenta con un sistema de administración de dispositivos que permite registrar dispositivos en grandes cantidades y organizarlos en grupos, para después agregar políticas de acceso para los dispositivos. Cada dispositivo debe contar con credenciales de acceso y todo el tráfico de datos debe ser encriptado por la capa de transporte seguro.
- Microsoft Azure para IoT.- Cuenta con un sistema de administración de dispositivos. Los dispositivos IoT son los responsables de iniciar la conexión a dicho sistema y usar la autenticación TLS con un certificado X.509, los cuales se validan con un registro dentro del sistema.
- Google Cloud IoT Core.- Cuenta con un sistema de administración de dispositivos, el cual incluye los procesos de registro, autentificación y autorización. Utiliza los protocolos MQTT y HTTP para la administración pública o privada utilizando JSON Web Tokens, así como también el uso de protocolos TLS para la comunicación.

Resultados

Para la comparación de rendimiento entre las tres plataformas se realizaron mediciones entre el tiempo que transcurre desde que el cliente produce y envía un mensaje hasta que este mensaje es recibido por un cliente o una aplicación.

Los resultados muestran cómo las plataformas, aunque con diferentes tiempos de servicio, responden de manera uniforme, garantizando así niveles de rendimiento predecibles que cumplan con las especificaciones indicadas.

Conclusiones

Se puede tomar para el proyecto las características en común que tienen las tres plataformas: las cuales son el IoT Hub, que proporciona una conexión con los dispositivos IoT, un gestor de reglas de comunicación para el manejo de datos a la plataforma, y la integración con otras aplicaciones existentes en cada plataforma.

3.2.5. Internet de las cosas: un estudio sobre arquitectura, tecnologías, protocolos y desafíos

Descripción general de la investigación

En esta investigación [12] se puede apreciar un panorama general sobre las tecnologías de recolección y comunicación de los datos entre dispositivos IoT, así mismo, muestra una descripción de alto nivel de la arquitectura en cinco capas, lo cual brinda un mayor entendimiento y se pueden extraer características que serían de utilidad en la arquitectura. Se describen también los desafíos, así como las tecnologías y protocolos que juegan un rol vital para el IoT.

Tecnologías utilizadas

El trabajo hace mucho énfasis en la arquitectura de cinco capas, pero menciona también las arquitecturas SOA y SOM. A continuación, se describen brevemente las cinco capas de la arquitectura.

- Percepción.-Es una capa de hardware y se encarga de recolectar la información del entorno físico, procesarla y transferirla a la siguiente capa.
- Abstracción.-Es la responsable de transferir la información desde la capa de percepción a través de tecnologías de red como lo son ZigBee, 3G, GSM, WiFi, infrarrojo, BLE, entre otras.
- Administración de servicios/Middleware.- Proporciona una interfaz entre los componentes de IoT y el sistema.
- Aplicación.- Proporciona varios servicios de aplicación según lo solicitado por el cliente y toda la interpretación de la información ocurre a este nivel.
- Administración/Negocio.- Realiza el monitoreo y la administración de las otras capas, administra todas las aplicaciones y servicios IoT y proporciona informes de análisis de alto nivel.

Se explican las tecnologías enfocadas a la adquisición y transmisión de datos, pero destacan más los protocolos para la recolección y transmisión de datos, así como la comunicación entre dispositivos y máquinas.

Resultados

En el trabajo se exponen los criterios de calidad que deben de estar presentes en los servicios para IoT, de los cuales son importantes la confiabilidad, movilidad, disponibilidad, escalabilidad, rendimiento e interoperabilidad.

Conclusiones

Utilizar en el proyecto una arquitectura de cinco capas puede proporcionar la modularidad requerida para el sistema a desarrollar, permitiendo mantener separadas las acciones dentro del sistema.

3.2.6. LISA: arquitectura ligera de servicios de IoT sensible al contexto

Descripción general de la investigación

En esta investigación [39] los autores proponen una arquitectura ligera sensible al contexto de servicios IoT llamada LISA. El propósito de la arquitectura LISA es filtrar y dirigir los servicios relevantes al usuario con base en el contexto del usuario, utilizando modelos de decisiones de usuario. El propósito es reducir la sobrecarga de datos entre el usuario y el servicio. Los autores presentan el diseño de la arquitectura, la cual permite que los servicios importantes de IoT alcancen al usuario con base en la información de contexto que utilizan.

Tecnologías utilizadas

Los módulos de la arquitectura LISA propuesta por los autores se describen a continuación:

- Módulo iniciador de consultas.- Este módulo desencadena una petición al generador de consultas automático con información de contexto del usuario (cambio de lugar, tiempo o requerimientos).
- Módulo generador de consultas.- Se encarga de realizar una consulta con información de contexto de cada usuario para poder obtener los servicios más apropiados.
- Proveedor de servicios web.- Recolecta la información de los servicios web que se ofertan de diversos proveedores y la envía al módulo de adaptación de servicios web, los cuales son seleccionados en función de la información de contexto y estos son ofertados al usuario.

- Módulo de adaptación de servicios web.- Selecciona los servicios verificando los datos recopilados de los proveedores de servicios con el modelo de usuario de IoT para conocer los servicios web más adecuados para dicho usuario, y posteriormente adapta el contenido recopilado para presentarlo al usuario de la mejor manera.
- Agente estático.- Administra el funcionamiento general del sistema y permite que funcione adecuadamente en un entorno distribuido mediante la generación y el envío de agentes móviles.
- Agente móvil.- Actúan en nombre del agente estático para la generación de consulta y realizar la adaptación y proveer el servicio.

Resultados

Los resultados se vieron reflejados en el caso de estudio, donde implementan la arquitectura en una aplicación turística que envía los servicios (salud, transportación, aprendizaje, museos y turismo) con base en los resultados de la consulta del contexto del usuario (registro de lugares, clima, medios de transportación, puntos turísticos e instituciones educativas). Los usuarios son clasificados en cuatro categorías dependiendo de su edad e ingresos económicos.

Conclusiones

En este trabajo se puede apreciar la utilidad de brindar servicios específicos mediante la utilización de información de contexto del usuario, lo cual ayuda a reducir la cantidad de datos utilizados para la comunicación, así mismo da opciones que sean relevantes para el usuario en el lugar y tiempo que se encuentre.

Este enfoque de envío de servicios puede ser de utilidad para trabajos futuros, donde se ofrecen una mayor cantidad de servicios de manera simultánea para distintos tipos de usuario.

3.2.7. Un marco ligero para soluciones inteligentes de IoT

Descripción general de la investigación

En esta investigación [40] los autores proponen una arquitectura ligera para soluciones inteligentes de IoT utilizando los componentes de software libre de FIWARE. El propósito de la investigación es la de generar un sistema de localización en interiores mediante el uso

de tecnologías WiFi y BLE. Los autores buscan promover el uso de software libre para lograr desarrollos a la medida de acuerdo con las necesidades de cada proyecto IoT.

Tecnologías utilizadas

El framework propuesto para el desarrollo de aplicaciones IoT consta de cuatro capas:

- Capa de adquisición de datos.- Consiste en una aplicación móvil encargada de la adquisición de datos de los objetos de su entorno independientemente del tipo de dispositivo y del protocolo de comunicación que se utilice.
- Capa de procesamiento de datos.- Se compone de múltiples módulos que se comunican mediante servicios REST a través del protocolo HTTP. El objetivo de esta capa es la de administrar y mantener la integridad de los datos que serán mandados a la siguiente capa.
- Capa de administración de datos.- Es responsable de almacenar la información capturada en las capas anteriores, así como de distribuir la información de la capa de procesamiento a la capa de aplicación.
- Capa de aplicación.- Se encarga de la representación gráfica de los datos obtenidos y procesados a través de una interfaz que sea comprensible para el usuario, así como también la administración de los dispositivos en el sistema.

Resultados

Los resultados se vieron reflejados en el caso de estudio, donde implementaron el framework para el desarrollo de una aplicación para localizar personas en interiores mediante el cálculo de la intensidad de señales de los dispositivos WiFi y BLE que se encuentran dentro del edificio.

El sistema fue desarrollado mediante el uso de la arquitectura mencionada y se adaptaron los componentes a las herramientas que ofrece FIWARE. La arquitectura en combinación con los componentes de FIWARE permitió el desarrollo ágil y de bajo costo del sistema IoT para la localización de personas en espacios cerrados.

Conclusiones

En este trabajo se puede apreciar cómo dividir los componentes en una arquitectura de múltiples capas permite la integración de herramientas de código abierto que a su vez facilitan

el desarrollo y la comunicación entre las capas. La forma de comunicación entre las capas es de gran utilidad, ya que ayudan a la estandarización de los protocolos de comunicación entre los diversos dispositivos. Considero que el modelo vista controlador utilizado en la capa de aplicación es de mucha utilidad para el desarrollo, aunque esto puede ser muy extenso, y que se necesita un modelo, una vista y el controlador para cada categoría de dispositivo a utilizar.

3.2.8. Comparación de la localización en interiores basada en la intensidad de señales para edificios inteligentes con internet de las cosas

Descripción general de la investigación

En esta investigación [41] se atiende el problema de la precisión y la eficiencia para localizar dispositivos en espacios cerrados. Debido a que el GPS no es una opción para resolver este problema, los autores utilizaron otras tecnologías inalámbricas capaces de localizar al dispositivo en estos ambientes. En este trabajo los autores comparan tres de las tecnologías más comunes para la localización en espacios cerrados (el protocolo Zigbee, así como de dispositivos emisores Bluetooth Low Energy y WiFi). La comparación es con base en la exactitud de la localización y el consumo de energía.

Tecnologías utilizadas

Los autores efectuaron tres experimentos: 1) con tres Raspberry Pi 3 modelo B como emisores de señales WiFi, 2) con tres *beacons* Gimbal serie 10 emisores de señales Bluetooth y 3) utilizando tres Arduino Uno serie 2 con tres Xbee emisores de señales Zigbee. En los tres casos se utilizó el algoritmo de trilateración para determinar la localización de un usuario con base en la intensidad de la señal recibida de las distintas tecnologías.

Resultados

Los resultados obtenidos por los autores al realizar los tres experimentos bajo las mismas condiciones fueron los siguientes. En el experimento 1) se obtuvo un margen de incertidumbre de 0.5183 metros. En el experimento 2) se obtuvo un margen de incertidumbre de 1.1143 metros. Y, por último, en el experimento 3) se obtuvo un margen de incertidumbre de 5.1317 metros.

Conclusiones

Para el desarrollo del proyecto, concretamente para el cálculo de la posición, se deben tomar en cuenta el uso de señales WiFi, así como BLE, se debe buscar un balance entre precisión y consumo energético.

3.2.9. Un breve estudio de conceptos de arquitectura de software y arquitectura orientada a servicios

Descripción general de la investigación

En esta investigación [42] se muestran los principales conceptos y características de las arquitecturas orientadas a servicios, el rol principal de una arquitectura de este tipo, los beneficios, características y los estándares en los que se basa y los impedimentos más comunes para una correcta implementación.

Tecnologías utilizadas

Los autores analizan las cualidades de la Arquitectura de Software, en particular de la Arquitectura Orientada a Servicios. Se utiliza el estándar para sitios web denominado Lenguaje de Descripción de Servicios Web (WSDL, por sus siglas en inglés) que abarca los aspectos básicos de la descripción de un servicio.

Resultados

Los resultados de la investigación nos brindan una idea del rol que juegan las arquitecturas de software en el proceso de desarrollo. Se define SOA como "Un paradigma para organizar y utilizar las capacidades distribuidas que pueden estar bajo el control de diferentes dominios de propiedad. Proporciona un medio uniforme para ofrecer, distribuir, interactuar y usar capacidades para producir efectos deseados consistentes con las condiciones y expectativas medibles*"*[43]

Conclusiones

Se considerará para el proyecto el uso de una arquitectura orientada a servicios debido a que presenta características destacables que se buscan en este proyecto, como la modularidad, la interoperabilidad y la baja o nula dependencia en los servicios.

3.2.10. Un modelo de identificación de servicios para la arquitectura orientada a servicios

Descripción general de la investigación

En esta investigación [44] los autores proponen un modelo para mejorar la velocidad de búsqueda de servicios. Como caso de estudio, los autores utilizan un proyecto con problemas de tiempo de respuesta, envíos equivocados y eficiencia menor a la esperada. El modelo implementado permite identificar, clasificar y eliminar servicios.

Tecnologías utilizadas

El modelo propuesto por los autores para poder identificar servicios consiste en cuatro fases:

- Identificación de servicios similares basados en funcionalidad.
- Creación de bloques de servicios con un comportamiento similar.
- Los grupos de servicios diferentes se mantienen en bloques diferentes.
- Eliminación de servicios duplicados.

Resultados

Los resultados obtenidos por los autores muestran una mayor eficiencia en comparación al otro modelo, esto se logra al reducir el tiempo de búsqueda y a su vez mejorando el porcentaje de disponibilidad de los servicios en el modelo.

Conclusiones

La organización y clasificación de servicios por similitud de comportamiento y funcionalidad a largo plazo puede ser de beneficio para identificar servicios de reemplazo o modificación en casos donde la cantidad de servicios disponibles sea significativa, y a su vez reducir el tiempo de búsqueda para proveer el servicio deseado.

3.2.11. Estado de la práctica

A continuación, se presenta el análisis de algunas de las plataformas comerciales para la localización en espacios cerrados que existen en la actualidad. Se consideró de interés porque permitirá orientar el diseño de la arquitectura de servicios de esta propuesta de tesis.

3.2.11.1. Situm

Descripción general de la investigación

En el estudio realizado [45] se analizaron los servicios que proporciona esta plataforma, el tipo de tecnología, el proceso de calibración y la utilización de la tecnología.

Tecnologías utilizadas

La plataforma utiliza emisores de señales WiFi, Bluetooth y la lectura de campos magnéticos de las edificaciones. Emplea los siguientes sensores: acelerómetro, giroscopio, y podómetro de teléfonos inteligentes con sistema operativo Android o IOS, así como la aplicación Situm Mapping Tool.

Resultados

Se realizó un análisis a los servicios proporcionados por la plataforma para el posicionamiento en tiempo real, la creación y calibración de edificios, el servicio de análisis de trayectorias de los usuarios, así como las limitaciones y posibles mejoras que se le pueden implementar, desde la calibración del edificio hasta la experiencia de usuario.

Se destaca que las rutas no son generadas por sistema, las debe establecer primero un administrador para poder usarlas. Del análisis realizado se puede obtener una base para la creación de aplicaciones, así como puntos clave a tratar para realizar mejoras en la experiencia de usuario durante los procesos de calibración que estos pueden llegar a ser tediosos.

Conclusiones

La plataforma ofrece soluciones de posicionamiento en espacios cerrados en unas pocas horas, con la mínima infraestructura y una precisión de 1 a 5 metros, sin la necesidad de que el teléfono inteligente cuente con conectividad de manera continua.

Lograron implementar un *Software as a Services* que se encuentra disponible en modo *On-Premise* para desarrollar y gestionar soluciones de localización para guiado de

pasajeros, visitantes y clientes, y para rastrear y monitorizar equipos de trabajo y activos móviles.

3.2.11.2. Mapwize

Descripción general de la investigación

En el estudio realizado a la plataforma [46] se analizaron los servicios que proporciona, el tipo y el uso de la tecnología y el proceso de inicialización.

Tecnologías utilizadas

Se efectuó el proceso de inicialización en donde se usaron las siguientes tecnologías para las pruebas físicas, emisores de señal WiFi, Bluetooth y campos magnéticos existentes, un teléfono inteligente con sistema operativo Android con los siguientes sensores: acelerómetro, giroscopio y podómetro, en conjunto con la aplicación Mapwize. Cabe destacar que la plataforma funciona con múltiples tecnologías de múltiples sistemas de posicionamiento en interiores.

Resultados

En el análisis realizado al servicio que permite inicializar un nuevo edificio, el cual consiste en delimitar en el mapa el área que cubre el edificio, después permite agregar los planos de distintos niveles que pueda tener el edificio, para después proceder con el diseño de los nodos para la navegación entre las diversas subáreas del edificio.

Conclusiones

Esta plataforma ofrece servicios muy similares a los de la plataforma Situm, pero cabe destacar un servicio que permite tener múltiples “universos”, los cuales son vistas del plano, que dependen del tipo de usuario que lo accede, por ejemplo, a un usuario de tipo visitante solo se le muestra la información relevante, lo cual sería muy distinto para un usuario de tipo mantenimiento, es decir depende del contexto del usuario para poder desplegar la información. Todo esto da una noción de posibles servicios para agregar al sistema.

3.2.11.3. Indoora

Descripción general de la investigación

En el estudio realizado a la plataforma [47] se analizaron los servicios que proporciona, el tipo y el uso de la tecnología y el proceso de inicialización.

Tecnologías utilizadas

La plataforma trabaja solo con señales WiFi y Bluetooth, de momento las pruebas están limitadas solo a dispositivos con sistema operativo iOS, pero la plataforma es funcional también en dispositivos con sistema Android.

Resultados

La plataforma ofrece una precisión de 2 metros en promedio, este nivel de precisión se mantiene aún en condiciones menos favorables como los edificios con techos a 20 metros de altura. Utiliza hasta seis veces menos *Beacons* en comparación con otros competidores para poder ofrecer el mismo nivel de precisión. La localización es compatible con cualquier marca de emisores BLE, gracias a su algoritmo patentado para el posicionamiento.

Conclusiones

Esta plataforma ofrece soluciones de posicionamiento en espacios cerrados en unas pocas horas, con la mínima infraestructura y reduciendo el uso de emisores BLE y aun así lograr una precisión promedio de 2 metros. Se trata de una plataforma que se encuentra disponible para desarrollar y gestionar soluciones de localización, proporcionando servicios similares a sus competidores, pero reduciendo el número de tecnologías utilizadas.

3.2.11.4. Navisens

Descripción general de la investigación

En el estudio realizado a la plataforma [48] se analizó la tecnología denominada motionDNA, el proceso de calibración inicial y cómo utilizar la tecnología.

Tecnologías utilizadas

La tecnología de motionDNA emplea los sensores inerciales (acelerómetro, giroscopio y podómetro) de los dispositivos móviles con sistema operativo Android e iOS, así como la aplicación correspondiente de Navisens Maps.

Resultados

Se analizaron los servicios proporcionados por la plataforma para el posicionamiento en tiempo real y la calibración de la aplicación. Ofrece un proceso de calibración sencillo pero eficaz, el cual consiste en seleccionar la posición y orientación actual, a partir de eso el sistema ya permite realizar un seguimiento muy preciso en dos o tres dimensiones.

Conclusiones

Esta plataforma ofrece una tecnología capaz de funcionar en ambientes abiertos o cerrados, ofreciendo una precisión menor a dos metros, sin la necesidad de la dependencia de señales WiFi y Bluetooth. Todo esto brinda un aporte significativo debido a que en cuestión de precisión el agregar más datos a la fórmula permitirá obtener un mejor resultado.

3.2.11.5. Litum IoT

Descripción general de la investigación

En el estudio realizado a la plataforma [49] se analizaron los servicios que proporciona, el tipo y el uso de la tecnología.

Tecnologías utilizadas

La tecnología que utiliza son las etiquetas UWB-RFID recargables con botón de pánico y los respectivos lectores de señal ubicados en posiciones estratégicas dentro de los edificios. Ofreciendo una precisión menor a un metro de distancia.

Resultados

Se analizaron los servicios proporcionados por la plataforma para el posicionamiento en tiempo real, vallas virtuales y el monitoreo en tiempo real del personal. La plataforma promete una precisión menor a un metro de distancia, con notificaciones en caso de emergencia o el acceso a áreas no autorizadas.

Conclusiones

Esta plataforma ofrece soluciones de posicionamiento en espacios cerrados en unas pocas horas, con la mínima infraestructura y una precisión de menos de un metro. En esta plataforma se descarta el uso de los teléfonos inteligentes, lo cual se reemplaza con dispositivos de mayor calidad y que están diseñados específicamente para la localización en interiores.

3.3. Observaciones de trabajos relacionados

En esta Sección se presentan los hallazgos más relevantes identificados en el estado del arte. Los hallazgos se enlistan a continuación de manera puntual con la finalidad de denotar las partes de mayor interés.

3.3.1. Arquitecturas

Las observaciones relevantes sobre arquitecturas son:

- Los trabajos [35], [37] presentan arquitecturas SOA o SOM para solucionar las necesidades de modularidad en la arquitectura, donde el middleware es el que proporciona la solución para la conectividad de dispositivos.
- En [36] se presenta una arquitectura compuesta que busca modificabilidad y se basa en el sistema de tres capas.
- Los trabajos [12], [40] presentan una arquitectura basada en capas que permite la modificabilidad en el sistema.
- [39], [40] Utilizan un framework ligero para implementar aplicaciones ágiles y a su vez [40] utiliza los componentes de la plataforma FIWARE para el desarrollo ágil.

3.3.2. Tecnologías de localización en interiores

Las observaciones relevantes sobre tecnologías son:

- En el trabajo [12] se muestran como las arquitecturas de 5 capas, como la SOA y SOM son compatibles con las tecnologías WiFi, BLE, Zigbee, entre otras.
- El trabajo [40] utiliza las señales WiFi y BLE para la localización de personas, y a su vez utiliza componentes de FIWARE para el desarrollo ágil.

- En [41] los autores realizaron un estudio de las tecnologías WiFi, BLE y Zigbee para determinar la que ofrece mayor precisión.

3.3.3. Evaluación de requerimientos

Las observaciones relevantes identificadas sobre evaluación de requerimientos:

- En la publicación [35] se presentan técnicas de diseño, implementación y validación para arquitecturas SOM.
- En [36] se muestra cómo evaluar la modificabilidad de una arquitectura de forma cuantitativa.
- Y en [38] se evalúa el desempeño de tres plataformas comerciales donde se puede apreciar las técnicas y la evaluación en estas.

3.4. Comparación de trabajos relacionados

En esta Sección, se presentan tres comparaciones de los trabajos relacionados con la investigación actual. En la primera se comparan los trabajos refiriéndose a la arquitectura, en la segunda se comparan los trabajos mediante las tecnologías y en la tercera se comparan las técnicas de evaluación de requerimientos.

3.4.1. Arquitecturas

En la Tabla 2 se presenta una comparación entre los trabajos identificados que mencionan la arquitectura utilizada y la presente investigación. Con la finalidad de identificar el patrón y el porqué de la elección, en esta comparación se establecieron los siguientes criterios:

1. Describe una arquitectura enfocada al IoT.
2. El diseño busca solventar un atributo de calidad específico.
3. Funcionamiento con múltiples tecnologías de localización.
4. Contexto teórico (aborda los aspectos necesarios para entender las arquitecturas).
5. Diseño de la arquitectura usando como base el patrón de capas.
6. Permite el desarrollo ágil al utilizar componentes de otras plataformas.

Tabla 2 Comparación de trabajos sobre Arquitecturas de Software

Estudio	1	2	3	4	5	6
Rendimiento y desafíos de la arquitectura orientada a servicios para redes de sensores inalámbricos [35]	●	●		●		
Un enfoque cuantitativo para analizar la modificabilidad en el diseño arquitectónico de software de sistemas de aplicaciones ágiles [36]		●		●	●	
Investigación sobre la arquitectura general del middleware de Internet de las cosas para parques industriales inteligentes [37]	●	●	●	●	●	
Soluciones de Amazon, Google y Microsoft para IoT: arquitecturas y una comparación de rendimiento [38]	●		●	●		●
Internet de las cosas: un estudio sobre arquitectura, tecnologías, protocolos y desafíos [12]	●	●	●	●	●	
LISA: arquitectura ligera de servicios de IoT sensible al contexto [39]	●	●		●		
Un marco ligero para soluciones inteligentes de IoT [40]	●	●	●	●	●	
Un breve estudio de conceptos de arquitectura de software y arquitectura orientada a servicios [42]	●			●		
Un modelo de identificación de servicios para la arquitectura orientada a servicios [44]		●		●		
Presente investigación	●	●	●	●	●	●

3.4.2. Tecnologías

En la Tabla 3 se presenta una comparación entre los trabajos identificados haciendo énfasis en las tecnologías utilizadas. Para identificar las partes relevantes de las publicaciones en cuestión de tecnología, en esta comparación se establecieron los siguientes criterios:

1. Realiza una comparación de las tecnologías de localización.
2. Contexto teórico (detalla los conceptos tecnologías de localización).
3. Funcionamiento con los componentes de una o más plataformas Open Source.

Tabla 3 Comparación de trabajos sobre tecnologías inalámbricas

Estudio	1	2	3
Internet de las cosas: un estudio sobre arquitectura, tecnologías, protocolos y desafíos [12]		●	
Un marco ligero para soluciones inteligentes de IoT [40]		●	●
Comparación de la localización en interiores basada en la intensidad de señales para edificios inteligentes con internet de las cosas [41]	●	●	
Presente investigación		●	●

3.4.3. Evaluación de Requerimientos

En este apartado, se presenta una comparación entre los trabajos identificados sobre la evaluación de requerimientos que se muestran en la Tabla 4. Con la finalidad de identificar las partes relevantes se establecieron los siguientes criterios:

1. Describe de manera explícita la forma de evaluar los requerimientos.
2. Compara arquitecturas en función de atributos de calidad
3. Evaluación cuantitativa del requerimiento

Tabla 4 Comparación de trabajos sobre evaluación de requerimientos

Estudio	1	2	3
Rendimiento y desafíos de la arquitectura orientada a servicios para redes de sensores inalámbricos [35]		●	
Un enfoque cuantitativo para analizar la modificabilidad en el diseño arquitectónico de software de sistemas de aplicaciones ágiles [36]	●		●
Soluciones de Amazon, Google y Microsoft para IoT: arquitecturas y una comparación de rendimiento [38]	●		●
Presente investigación	●	●	●

Capítulo 4

Diseño y desarrollo de la arquitectura

4. Diseño y desarrollo de la arquitectura

En este capítulo se describe a mayor detalle la arquitectura propuesta para este trabajo, se describen los componentes principales y cómo éstos satisfacen las necesidades de modificabilidad del trabajo.

4.1. Diseño de la arquitectura

Para el diseño de la arquitectura se optó por un diseño compuesto utilizando el patrón en capas como base [11]. Este tiene la característica principal de poder actualizar libremente cada porción del sistema, brindando así flexibilidad y modificabilidad independiente de los elementos. Además, de acuerdo con [51], al implementar mecanismos de comunicación estandarizados entre las capas se logra dar una funcionalidad similar a la arquitectura orientada a servicios, lo cual a su vez otorga las ventajas asociadas de flexibilidad y agilidad para el desarrollo y mantenimiento del sistema. El patrón en capas proporciona una estructura estable para el desarrollo inicial de proyectos, ya que permite segmentar el sistema para posibles modificaciones, establecer mecanismos de comunicación estandarizados, así como la integración de servicios de una manera más simple. La arquitectura propuesta facilita la interacción con distintas plataformas de IoT, en nuestro caso de estudio, para ofrecer servicios de posicionamiento en espacios cerrados que utilizan diversas tecnologías y algoritmos para el cálculo de posición, logrando así un alto grado de intercompatibilidad entre componentes.

La modularidad que brinda el patrón de capas permite la modificabilidad y portabilidad de los componentes desarrollados, y a su vez que estos componentes puedan ser distribuidos en distintos servidores para así reducir la carga de procesamiento y almacenamiento de datos. En nuestro caso de estudio, la arquitectura propuesta permite realizar modificaciones rápidas para adaptar los componentes a los servicios que ofrecen las plataformas IoT, FIWARE y DeviceHive. Asimismo, el funcionamiento resultó muy similar independientemente de la plataforma utilizada, pudiendo evaluar la característica de modificabilidad de la arquitectura en las dos plataformas IoT.

Para dar solución a la problemática de esta investigación se diseñó una arquitectura capaz de proveer servicios de localización en interiores, y a su vez mantener la modificabilidad dentro del sistema. En la Figura 10 se muestra una vista general de la

arquitectura, la cual cuenta con cuatro capas principales (color azul), los servicios de cada capa se encuentran en color amarillo, donde las funciones de los servicios se encuentran en recuadros color blanco y la comunicación entre capas se denota como una línea punteada entre las capas para la comunicación REST. En cada capa se puede incrementar el número de servicios para ampliar la funcionalidad.

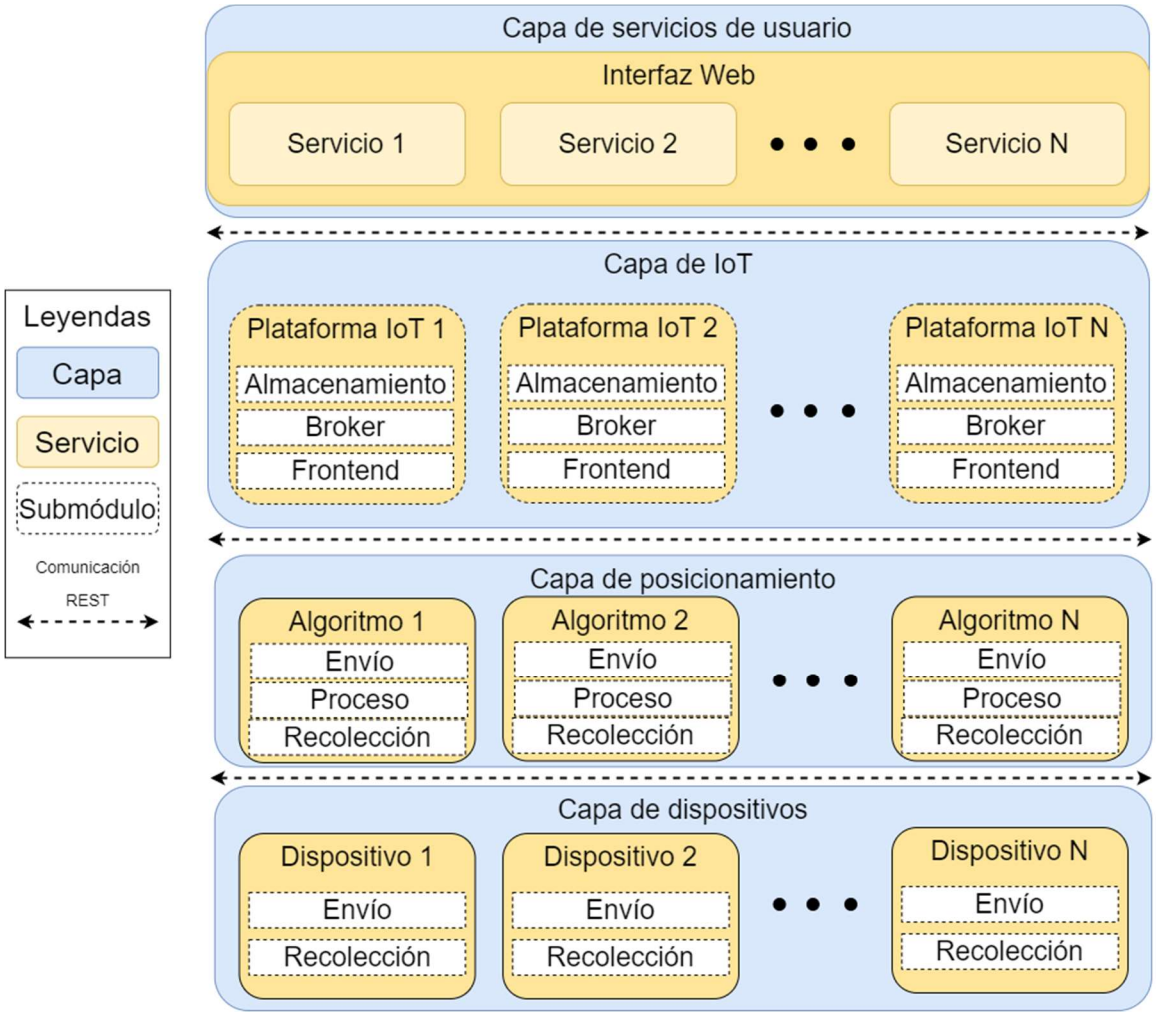

Figura 10 Diseño de la arquitectura

El diseño presentado en la Figura 10 muestra la vista general de la arquitectura, la capa de servicios de usuario presenta al usuario una interfaz a través de la cual puede acceder a los servicios proporcionados de una forma gráfica y entendible para cualquier usuario. En esta capa se encuentran los servicios finales para desplegar información de posicionamiento al usuario. La capa está diseñada específicamente para la integración y modificación ágil de servicios que utilicen la información de posicionamiento en interiores.

Después de la capa de servicios de usuario se encuentra la capa de IoT, la cual mediante protocolos de comunicación REST envía la información de posicionamiento a la capa de aplicación, y a su vez recibe y almacena la información proporcionada por la capa de posicionamiento. Está diseñada para funcionar como intermediario entre los servicios desarrollados y los componentes de las plataformas IoT como FIWARE y DeviceHive, no obstante, se puede utilizar cualquier plataforma que utilice un esquema de publicación/suscripción de datos que utilice el protocolo REST. En esta capa es obligatorio la existencia de al menos una plataforma para el funcionamiento.

Inmediatamente se encuentra la capa de posicionamiento, la cual se dedica a la recolección de datos que son enviados por dispositivos móviles de la persona a monitorizar, para posteriormente realizar los cálculos necesarios (dependiendo del algoritmo con el que sea implementado) para el cálculo de la ubicación y enviar estos datos a la capa de IoT. Para fines prácticos de nuestro caso de estudio, se utilizaron módulos de simulación de movimientos de personas.

Y, por último, la capa de dispositivo se encarga de la recolección de señales inalámbricas (WiFi y BLE) del entorno y envía la intensidad de cada señal a la capa de posicionamiento para su procesamiento y así poder obtener la ubicación en el espacio.

4.1.1. Componentes principales

Los componentes que se encuentran en las capas de servicios de usuario, IoT y de posicionamiento forman parte fundamental del proyecto y por lo tanto, requieren una explicación más detallada en cuestiones de funcionamiento y tecnología utilizada.

En la capa de servicios de usuario se encuentra la interfaz principal, como se mencionó anteriormente, permite el consumo de servicios de una forma más amigable para el usuario que solo el consumo de datos crudos que se pueden obtener directamente de la capa del middleware. Se puede agregar cualquier número de servicios, siempre y cuando utilicen el mismo formato de comunicación. Los servicios utilizan tecnologías utilizadas en aplicaciones web, como PHP, JavaScript y consultas REST. Cada servicio se implementa de forma modular para poder estandarizar su desarrollo, así como también las futuras modificaciones que se requieran y la habilitación de los servicios dependiendo los privilegios de los usuarios.

En la capa del IoT es de vital importancia la forma estandarizada de comunicación para así asegurar el correcto envío de datos de los componentes de las plataformas de IoT con esquemas de publicación/suscripción que se requieran utilizar (Dependiendo de las necesidades del usuario). Esta ca capa permite integrar de forma fácil componentes de las plataformas IoT aún incluso si estos no se encuentran en el mismo servidor, permitiendo tener múltiples instancias de las plataformas. Para demostrar las bondades que proporciona la capa, se implementan los componentes de la plataforma FIWARE en un servidor privado en la nube. Para los componentes de la plataforma DeviceHive se utilizó directamente la infraestructura que proporciona esta plataforma, todo esto mientras el resto del sistema se encuentra en un servidor local que consume los servicios de ambas plataformas.

Y, por último, la capa de posicionamiento permite la integración de los servicios de posicionamiento que transforman las señales inalámbricas recolectadas por los dispositivos móviles y las transforman mediante algoritmos en coordenadas relativas a los planos de un edificio específico y transfieran la información en un formato específico para la comunicación con la capa del middleware. Considerando que los servicios de posicionamiento no entran dentro de las limitaciones del proyecto, se utilizaron simuladores de posición que envían los datos de coordenadas relativas de los usuarios a las plataformas de IoT.

4.2. Implementación de la arquitectura

En esta sección se describe el proceso para implementar la arquitectura capa por capa, mostrando las herramientas, lenguajes y plataformas utilizadas para el desarrollo de este proyecto con el fin de servir como guía para replicar el proyecto, sin embargo, no son estrictamente necesarias para un desarrollo similar. Algunas de las herramientas descritas se utilizan sólo para fines prácticos. La implementación mostrada es una instancia del diseño de la arquitectura mostrada en la Figura 10, donde se implementan dos servicios de localización en la capa de servicios de usuario, una plataforma en la capa de IoT (FIWARE), y un servicio de simulación en la capa de posicionamiento con la finalidad de agilizar la implementación, por lo tanto, no se utiliza la capa de dispositivo, la implementación se realizó de acuerdo con la Figura 22 presentada en la sección 5.2 utilizando los componentes de FIWARE.

El código completo para cada servicio mostrado en esta sección del documento puede ser consultado en este enlace al repositorio de GitHub https://github.com/ErickInCo/Tesis-Maestria.

4.2.1. Implementación de la capa de servicios de usuario

Para implementar esta capa es necesario contar con un servidor de PHP para la ejecución de los archivos, para este trabajo se utilizó un servidor local con sistema operativo Windows 10 ejecutando los componentes de WampServer 3.1. La interfaz WEB presentada en la Figura 11 muestra la ejecución de dos servicios de localización, del lado izquierdo se muestra una vista de la posición de los usuarios en un mapa, de lado derecho se muestra la información correspondiente para cada usuario con su respectiva alerta, mientras que en la Figura 12 se presenta fragmentos del código en HTML para generar la interfaz, en la cual se instancian los servicios de mapa y alertas en elementos iframe, y se configura una actualización continua para cada iframe para mantener los datos de los servicios en actualización.

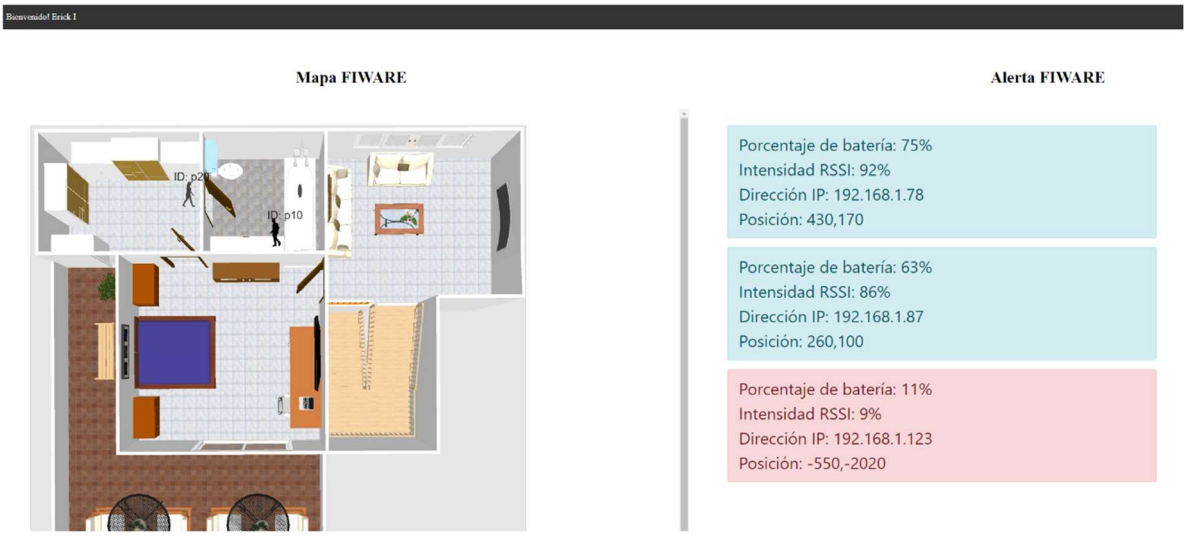

Figura 11 Interfaz web con dos servicios

```html
<html>
<head></head>
<style></style>
<body>

    <!-- Contenedor de iframes para servicios-->
    <div class="grid-container">
        <div class="grid-item">
            <h4 align="center">Mapa FIWARE</h4>
            <iframe id="iframe" src="\ProyectoLuz\mapaFW.php" style="border:none;height:850px; width: 100%" ></iframe>
        </div>
        <div class="grid-item" style="height:auto;">
            <h4 align="center">Alerta FIWARE</h4>
            <iframe id="iframe2" src="\ProyectoLuz\alertaFW.php" style="border:none;height:850px; width: 100%" ></iframe>
        </div>

    </div>
    <!-- Script para actualizar datos de los iframes-->
    <script>
        window.setInterval(function() {
            reloadIFrame()
        }, 1000);

        function reloadIFrame() {
            console.log('reloading..');
            document.getElementById('iframe').contentWindow.location.reload();
            document.getElementById('iframe2').contentWindow.location.reload();
        }
    </script>
</body>
</html>
```

Figura 12 Archivo HTML para interfaz web

El primer servicio implementado en esta capa es el servicio de visualización de posición en un mapa del edificio, donde se obtienen los datos de la capa de IoT para ser mostrados al usuario de forma gráfica como se muestra en la Figura 13 donde se visualizan dos entidades en el edificio.

Figura 13 Servicio de mapa

En la Figura 14 se muestran fragmentos del código del servicio donde se realiza la conexión con la plataforma FIWARE mediante consulta REST, se limpian los datos de caracteres no deseados y se almacena en variables para su uso mediante una función en JavaScript con PHP que tiene la función de mostrar el indicador de posición en el mapa por cada dispositivo.

El segundo servicio implementado, el servicio de alerta, consiste en mostrar información del modelo de datos en componentes visuales, donde se notifica al usuario mediante el cambio del color del componente si se aleja de la zona de monitoreo, pasando de un color azul a uno rojo, La Figura 15 muestra los datos de tres entidades, donde las primeras dos (p10 y p20) se encuentran en la zona de monitoreo, mientras que la entidad p30 ya se encuentra fuera de la zona, por lo cual muestra en color rojo el recuadro con la información. El fragmento de código para implementar el servicio de alertas se muestra en la Figura 16.

```php
<?php
// Consultar datos de la plataforma IoT mediante REST
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, "http://IPFIWARE:1026/v2/entities/");
$response = curl_exec($ch);
curl_close($ch);

// Limpieza de caracteres
$myArray = explode(',', $rlimpio);

//Almacenamiento de variables
$p=0;
for ($i = 0; $i < count($myArray); $i=$i+6) {
    datos[$i]=$myArray; ...
}
?>

<html>
<head></head>
<style></style>
<body>
    <!-- Cargar imagenes mapa e indicadores-->
    <img id="m" width="700" height="400" src="\ProyectoLuz\img\mapaV2.jpg" > ...
    <!-- Inicializar elemento canvas para mostrar las imagenes-->
    <canvas align="top" id="myCanvas" width="900" height="800"></canvas>

    <!-- Script para cargar la posicion de las personas-->
    <script type="text/javascript">
    // Funcion para cargar datos en mapa
    window.onload = function() {
        // Inicializar variables
        var canvas = document.getElementById("myCanvas"); ...
        // Dibujar mapa
        ctx.drawImage(img, 0, 0);
        <?php
            $p=0;
            for ($i = 0; $i < count($myArray); $i=$i+6) {
                // Dibujar indicadores de posicion
                echo "ctx.drawImage(img3,".$px[$p].",".$py[$p].", 30, 50);"; ...
                // Colocar indicador de nombre
                echo "ctx.fillText('ID: p".(string)((int)$p+1)."0', ".$px[$p].", ".$py[$p].");";
                $p=$p+1;
            }
        ?>
    };
    </script>
</body>
</html>
```

Figura 14 Fragmento de código PHP para servicio de visualización

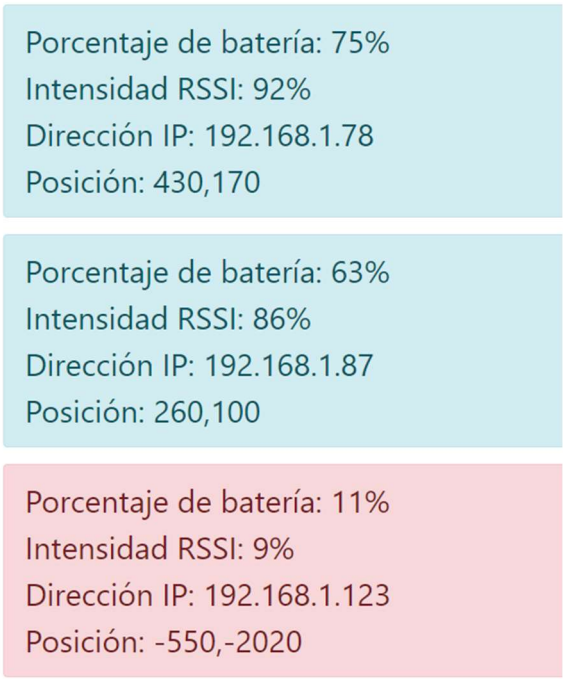

Figura 15 Servicio de alertas

```php
<?php
// Consultar datos de la plataforma IoT mediante REST
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, "http://IPFIWARE:1026/v2/entities/");
$response = curl_exec($ch);
curl_close($ch);

// Limpieza de caracteres no desados
$myArray = explode(',', $rlimpio);

// Almacenamiento de variables
$p=0;
for ($i = 0; $i < count($myArray); $i=$i+6) {
    datos[$i]=$myArray;...
}
?>
<html>
<head></head>
<style></style>
<body>
    <div class="grid-container">
        <div class="grid-item">
            <?php
                // Creacion de componentes con datos y configurar la alerta de acuerdo a la distancia
                $p=0;
                for ($i = 0; $i < count($myArray); $i=$i+6) {
                    echo "<div class='alert alert-info'>";...
                }
            ?>
        </div>
    </div>
</body>
</html>
```

Figura 16 Fragmento de código PHP para servicio de alerta

Los servicios web de visualización y alertas son implementados para dar funcionalidad a los servicios en la capa de servicios de usuario se desarrollaron con las siguientes herramientas, para la maquetación inicial de la interfaz web se utilizó Adobe Dreamweaver 2021, el cual proporcionó la base del código HTML para la página principal, posteriormente con Visual Studio Code 1.69 (licencia educativa) y Sublime Text 3 para el desarrollo de las funciones de PHP y JavaScript de los servicios de localización y alertas y por último se montó en un servidor local con WampServer 3.1 que incluye el servidor Apache 2.4, el lenguaje PHP 7.2 y el Sistema Manejador de Base de Datos MySQL 5.7.

4.2.2. Implementación de la capa de IoT

Para implementar esta capa es necesario contar con un servidor capaz de ejecutar Docker, en el Anexo B se muestra como configurar un servidor Linux para instalar los componentes necesarios para Docker, así como también el despliegue de los componentes de FIWARE usando los archivos de configuración YML que se encuentran en el repositorio de GitHub (https://github.com/ErickInCo/Tesis-Maestria/tree/main/Capa%20de%20IoT).

Una vez implementados los componentes de la plataforma IoT (FIWARE) y tener identificada la dirección IP del servidor, así como el puerto para el *orion contex broker* (Puerto 1026), ya se pueden realizar las conexiones con la capa servicios de usuario indicando en la petición REST a la IP y puerto del servidor, y la conexión a la capa de posicionamiento se realiza con una petición REST a la dirección IP junto con el puerto utilizado.

4.2.3. Implementación de la capa de posicionamiento

Para facilitar el envío de datos desde la capa de posicionamiento hacia la capa de IoT se implementó un script en Python encargado de enviar los datos de posición de personas utilizando caminos predefinidos para demostración de múltiples personas de forma simultánea utilizando múltiples instancias del código mostrado en la Figura 17 donde se muestra el fragmento de código para simular él envió de datos de posicionamiento donde toma como parámetros iniciales el ID del dispositivo, calcula una posición en el mapa y la envía mediante petición REST a la plataforma IoT indicada en el URL utilizando el modelo de datos NGSI como plantilla para incluir los datos en la petición.

```python
import time
import requests
import random

url = "http://$IPIoT:1026/v2/entities/$ID/attrs"

modeloNGSI="""{"Type": { "value": "Device".....
        ...
        ...
    """
direcciones[i][p]=[1,[12,32]]...

while 1==1:
    # Seleccionar destino
    destino= direcciones[random.randint(0,len(direcciones))]
    # Agregar datos al formato NGSI
    modeloNGSI(destino, id, rssi, ip, x, y)
    # Enviar peticion REST
    response = requests.put(url(id), data=modeloNGSI, headers=headers)
    # Actualizar posicion actual
    if response.status_code==204:
        anterior=actual
        actual=destino-1
    # Esperar 2 seg para el proximo envio
    time.sleep(2)
```

Figura 17 Fragmento de código Python para servicio de posicionamiento

4.2.4. Implementación de la capa de dispositivo

Con el uso de un servicio de simulación en la capa de posicionamiento, la capa de dispositivo no fue implementada, pero se detallan las instrucciones generales para generar una aplicación para Android que realice la recolección y envió de señales de WiFi y BLE a la capa de posicionamiento.

La aplicación debe tener los permisos necesarios en el archivo de AndroidManifest.xml para acceder a la información de WiFi y BLE. El primer paso es generar un bucle infinito para recopilar y enviar los datos, donde se utilizarán las clases *WifiManager* y *BluetoothAdapter* para obtener la información de las señales inalámbricas. Como segundo paso es crear un objeto JSON con los datos recopilados y la información del dispositivo. El tercer paso es enviar la petición REST a la IP del servicio de posicionamiento a utilizar y por último establecer el tiempo para volver a iniciar con el bucle para que se recolecte información de manera continua.

4.3. Mapeo FIWARE y DeviceHive

Para este apartado se describirán los elementos principales que permiten utilizar los componentes de las plataformas de FIWARE y DeviceHive para su uso en el proyecto, como

estos componentes contribuyen a la estructura principal y como son compatibles con el resto de los elementos del sistema.

La plataforma FIWARE consta de componentes que permiten conectar dispositivos y servicios, los cuales pueden implementarse de diversas formas según sea el tipo de aplicación. DeviceHive también cuenta con diversos componentes que permiten conectar dispositivos y servicios, ambas plataformas tienen como objetivo proporcionar herramientas para crear aplicaciones para dispositivos IoT.

Los elementos principales de estas plataformas de forma general incluyen:

- Una biblioteca con componentes de interfaz de usuario que permite crear interfaces de usuario de manera sencilla.
- Una capa de servicio que proporciona almacenamiento de datos y funciones de gestión de acceso.
- Una capa de mensajería que permite a los usuarios comunicarse entre sí mediante solicitudes en protocolo REST.

4.4. Flujo de datos

En este apartado se describe el flujo de datos en la arquitectura, tomando en cuenta el modelo da datos NGSI de la plataforma FIWARE como modelo de datos estandarizado para la comunicación entre capas.

4.4.1. Modelo de datos NGSI

La interfaz de servicios de próxima generación (NGSI, del inglés *Next Generation Service Interface*) es una interfaz de programación de aplicaciones (API) desarrollada por FIWARE para acceder a los datos de entidades y relaciones en un sistema de internet de las cosas. Es un sistema basado en el protocolo REST que utiliza los comandos HTTP (Get, Put, Post, Delete) para acceder a los datos que se encuentran representados en formato JSON y se pueda acceder a las entidades mediante una URL específica para cada entidad.

Para nuestro proyecto se decidió utilizar el modelo de datos inteligente de la categoría “Detección Inteligente” como base para la comunicación entre capas de la arquitectura por los siguientes motivos. En primer lugar, es un modelo estandarizado que facilita la interoperabilidad entre distintos sistemas. Además, tiene una estructura clara y bien definida,

lo que permite su fácil comprensión y utilización al momento de consumir y enviar datos. En la Figura 18 se muestra el modelo de datos utilizado para la comunicación con la plataforma FIWARE y en la Figura 19 se muestra el modelo adaptado para funcionar con la plataforma DeviceHive.

Los modelos de datos de la categoría “Detección Inteligente” son una herramienta esencial para el desarrollo de ciudades o entornos inteligentes. Los modelos ofrecen una estructura estandarizada para la recopilación, almacenamiento y análisis de datos de múltiples fuentes, los dispositivos que entran en esta categoría son hardware o software destinado a realizar tareas relacionadas a la medición o ejecución de acuerdo con el ambiente en el que se encuentran [50]. Con la introducción del IoT, los dispositivos conectados pueden transmitir la información de forma estandarizada a un modelo ya establecido.

La Figura 18 muestra el modelo de datos tipo Dispositivo (*Device*), para este trabajo en cuestión se utilizaron solo las propiedades necesarias del modelo estándar. Los datos almacenados son un identificador, tipo de datos (*Device*), categoría, nivel de batería, fecha de primera conexión, número de serie, indicador de la intensidad de señal (rssi), dirección IP, estado del dispositivo, posición del dispositivo y dueño.

```
{
    "id": "p10",
    "type": "Device",
    "category": {
        "value": ["celphone"]
    },
    "batteryLevel": {
        "value": 0.75
    },
    "dateFirstUsed": {
        "type": "DateTime",
        "value": "2023-01-05T11:00:00Z"
    },
    "serialNumber": {
        "value": "9845A"
    },
    "rssi": {
        "value": 0.86
    },
    "ipAddress": {
        "value": "192.168.1.78"
    },
    "deviceState": {
        "value": "ok"
    },
    "location": {
      "type": "Point",
      "value": [-150,-200]
    },
    "owner": {
        "value": 1
    }
}
```


Figura 18 Modelo de datos NGSI para FIWARE

La Figura 19 muestra el modelo adaptado del modelo de datos NGSI Dispositivo en la plataforma de DeviceHive para su correcto funcionamiento, el formato de datos se coloca en la propiedad data, y se agregan propiedades como id, identificador de red, nombre, tipo de dispositivo, y bloqueo de edición.

```
{
    "id": "aQlnNFNALjGu1tdXUzTuMsIUJpAQzWaGtIEU",
    "name": "p10",
    "data": {
      "id": "p10",
      "type": "Device",
      "category": {
        "value": [
          "celphone"
        ]
      },
      "batteryLevel": {
        "value": 0.75
      },
      "dateFirstUsed": {
        "type": "DateTime",
        "value": "2020-09-11T11:00:00Z"
      },
      "serialNumber": {
        "value": "9845A"
      },
      "rssi": {
        "value": 0.86
      },
      "ipAddress": {
        "value": "192.168.1.78"
      },
      "deviceState": {
        "value": "ok"
      },
      "location": {
        "type": "Point",
        "value": [
          220,
          220
        ]
      }
    },
    "networkId": 7438,
    "deviceTypeId": 1,
    "isBlocked": false
  }
```


Figura 19 Modelo de datos adaptado NGSI DeviceHive

4.4.2. Capa de dispositivo

En esta capa ocurre la parte primordial del funcionamiento, los dispositivos móviles recolectan los datos de señales inalámbricas (WiFi, BLE, etc.) y generan un primer objeto JSON con los datos de señales, la información del dispositivo (identificador, porcentaje de batería, dirección IP, etc.) para ser enviado por peticiones REST a la capa superior de

posicionamiento que se encarga de realizar los pasos necesarios para ubicar al dispositivo en el entorno en el que se encuentra.

4.4.3. Capa de posicionamiento

Una vez los datos recibidos en formato JSON, se realiza el cálculo pertinente de acuerdo con el algoritmo de localización implementado. En esta etapa los datos se transforman al formato NGSI específico para cada plataforma IoT con sus respectivos datos, y se envían por peticiones REST para agregar o actualizar los datos que se encuentran en las plataformas IoT donde son recibidos por el bróker de estas.

4.4.4. Capa de IoT

Dependiendo de la plataforma IoT implementada el flujo de datos cambia, en caso de FIWARE los datos son recibidos por parte del Orion Context Broker y envía los datos en formato NGSI al componente de QuantumLeap donde la información es almacenada y se encuentra disponible para su consumo inmediato. En el caso de DeviceHive los datos son recibidos por el bróker Kafka y los dirige al servicio de Backend de DeviceHive para el almacenamiento y consumo de datos.

4.4.5. Capa de servicios de usuario

Está es la etapa final de los datos, los servicios web disponibles realizan consultas REST a la capa de IoT donde reciben los datos en formato NGSI de las entidades consultadas, los datos se transforman en variables independientes y estos se presentan al usuario final de forma práctica de acuerdo con el servicio utilizado.

4.5. Modificabilidad en los servicios

En este apartado, se explica cómo realizar cambios en la arquitectura en las capas de Servicios de usuario, IoT y posicionamiento, con el objetivo de añadir nuevas funcionalidades o editar el comportamiento de los servicios que se ofrecen. Para cada capa descrita se muestra el proceso para habilitar o deshabilitar y modificar un servicio.

4.5.1. Servicios de usuario (WEB)

Los servicios de usuario están pensados para ser diseñados e implementados como componentes mediante el uso de iFrames en HTML. Teniendo esto en cuenta, el servicio debe de funcionar por separado de los demás componentes. Para el uso de otros datos no especificados en el modelo de datos utilizado se tendrá que modificar el servicio de usuario, así como también el servicio de posicionamiento para que adjunte los nuevos datos al modelo.

Para poder habilitar o deshabilitar un servicio basta con conectar o desconectar el elemento iFrame que se encuentra ligado al servicio en la interfaz general. Este proceso se puede realizar de distintas formas dependiendo de las necesidades de la aplicación a utilizar. Pueden mostrarse solo los servicios dependiendo del nivel de acceso del usuario para un enfoque automático, o de forma manual para el mantenimiento del servicio a nivel general.

Para poder realizar modificaciones a un servicio de usuario, es necesario tener acceso al archivo principal del componente (PHP, Java, Node.js, etc.), realizar los cambios necesarios (Ejemplo cambiar la dirección de la plataforma IoT, modificar los datos recibidos, o de funcionamiento general), realizar los cambios necesarios y guardar el archivo.

Para agregar un servicio nuevo a la capa se necesita modificar el archivo de interfaz general, donde se agregará el código correspondiente para un elemento iFrame ligado al servicio del nuevo componente en la posición deseada para el despliegue en pantalla.

4.5.2. Servicios de IoT

Los servicios IoT están pensados para un despliegue rápido mediante el uso de contenedores de Docker. Las plataformas FIWARE y DeviceHive cuentan con documentación detallada para el despliegue de sus componentes. En caso de necesitar modificar los puertos o componentes a utilizar en cada plataforma, es necesario hacer los ajustes pertinentes al archivo de configuración YML. La Figura 20 muestra un ejemplo de archivo de configuración YML utilizado para estas plataformas y volver a levantar los componentes con el archivo de despliegue modificado.

Es posible ligar otra plataforma IoT a esta arquitectura. Para habilitar o deshabilitar una plataforma es necesario modificar los servicios de posicionamiento y redirigirlos a otra plataforma. De igual manera, para el consumo de servicios de usuario se debe de indicar la plataforma de la cual se realizarán consultas a los datos. La plataforma nueva debe de ser

implementada en un servidor que cuente con una dirección IP que pueda accederse desde las capas inferiores y superiores, pudiendo ser esta una dirección local (para entornos privados) o pública (para despliegue en diversos servidores). Una vez desplegada la plataforma, es necesario modificar la IP de los servicios que van a consultar datos de dicha plataforma. En el caso de que el modelo de datos cambie, habrá que realizar los cambios necesarios en los servicios de usuario y posicionamiento.

4.5.3. Servicios de posicionamiento

Los servicios de posicionamiento están pensados para generar el formato de datos NGSI para las plataformas IoT y su consumo en los servicios de usuario, por lo cual se debe respetar el formato de datos NGSI de tipo dispositivo o de lo contrario realizar los cambios pertinentes en la capa de servicios de usuario.

Para poder habilitar un nuevo servicio de posicionamiento es cuestión de indicar a los dispositivos móviles de la capa de dispositivo la nueva dirección IP donde se ubica el servicio, y el envío de datos realizarlo mediante peticiones REST a una plataforma IoT activa.

Para deshabilitar el servicio es cuestión de negar las peticiones REST dirigidas al servicio a inhabilitar y así se deja deshabilitado el tiempo necesario, ya sea por cuestiones de mantenimiento o modificaciones al comportamiento del servicio.

```yaml
version: '3'
services:
  orion:
    image: fiware/orion:${ORION_VERSION:-2.0.0}
    ports:
      - "1026:1026"
    command: -logLevel DEBUG -noCache -dbhost mongo
    depends_on:
      - mongo
    healthcheck:
      test: ["CMD", "curl", "-f", "http://0.0.0.0:1026/version"]
      interval: 1m
      timeout: 10s
      retries: 3
  quantumleap:
    image: ${QL_IMAGE:-smartsdk/quantumleap}
    ports:
      - "8668:8668"
    depends_on:
      - mongo
      - orion
      - crate
    environment:
      - CRATE_HOST=${CRATE_HOST:-crate}
      - USE_GEOCODING=True
      - REDIS_HOST=redis
      - REDIS_PORT=6379
      - LOGLEVEL=DEBUG

  mongo:
    image: mongo:3.2.19
    ports:
      - "27017:27017"
    volumes:
      - mongodata:/data/db

  crate:
    image: crate:${CRATE_VERSION:-4.1.4}
    command: crate -Cauth.host_based.enabled=false

-Ccluster.name=democluster -Chttp.cors.enabled=true -Chttp.cors.allow-ori
gin="*"
    ports:
      # Admin UI
      - "4200:4200"
      # Transport protocol
      - "4300:4300"
    volumes:
      - cratedata:/data

  grafana:
    image: grafana/grafana
    ports:
      - "3000:3000"
    environment:
      - GF_INSTALL_PLUGINS=grafana-clock-panel,grafana-worldmap-panel
    depends_on:
      - crate

  redis:
    image: redis
    ports:
      - "6379:6379"
    volumes:
      - redisdata:/data

volumes:
  mongodata:
  cratedata:
  redisdata:

networks:
    default:
        driver_opts:
            com.docker.network.driver.mtu: ${DOCKER_MTU:-1400}
```

Figura 20 Archivo YML de FIWARE

Capítulo 5

Caso de estudio

5. Caso de estudio

Para poder realizar una evaluación a la arquitectura, se plantea el desarrollo de un caso de estudio a través del cual se podrá observar el funcionamiento de la arquitectura, así como también el nivel de modificabilidad que esta proporciona. El caso de estudio consiste en el desarrollo de una aplicación para el seguimiento en interiores de personas (mediante el uso de un dispositivo móvil). El proyecto tiene la finalidad a futuro que sea utilizado para el seguimiento de grandes números de personas en edificios, así como la navegación en lugares complejos.

La arquitectura está basada en el modelo de capas, permite la implementación de otras arquitecturas dentro de cada capa, permitiendo la integración de componentes externos para el funcionamiento de la aplicación a desarrollar. De los componentes externos se utilizan dos plataformas con características y funcionamiento similar de software libre (FIWARE y DeviceHive) y pueden ser utilizadas por cualquier persona, esto con la finalidad de que pueda ser replicable para fines de análisis de la arquitectura o la aplicación en general.

La aplicación se desarrolló en un entorno Web para facilitar la visualización desde cualquier dispositivo que cuente con un navegador compatible con las tecnologías utilizadas. Se utilizará PHP como lenguaje principal y el estándar REST para la comunicación entre las capas utilizando mensajes JSON con información estructurada. La aplicación presentará al usuario un mapa del edificio donde se mostrará la ubicación e información de personas que se encuentran dentro del edificio, desplegará alertas en formato de texto en el caso que algún usuario salga del área de monitoreo.

Los componentes tanto de FIWARE como DeviceHive serán alojados en distintos servidores a los de la aplicación principal con la finalidad de mostrar la escalabilidad de los componentes en cuestión de potencia, donde se puede realizar en cualquier capa que requiera mayores características de cómputo. La estructura principal de la aplicación se encuentra en un servidor local para facilitar las pruebas durante las etapas de desarrollo, pruebas e implementación.

Para mantener un entorno de evaluación similar en ambas plataformas es necesario trabajar primero cada una por separado, estableciendo un caso de estudio correspondiente FIWARE así como a DeviceHive, esto se logra realizando una simplificación de la arquitectura presentada en la Figura 10, donde solo se utilizará una sola plataforma de IoT

para el manejo de los datos de posición, con la finalidad de reducir las variables que puedan alterar los datos de la evaluación de la modificabilidad existente en la arquitectura, una vez que se comprueben los valores, se realizó a la evaluación de forma simultánea utilizando ambas plataformas IoT en la versión final de ésta.

En cuestión de evaluar los resultados, se realizará de acuerdo con el método presentado por [36] mostrado en el capítulo 3.2.2 que se basa en diversas técnicas de evaluación de arquitecturas como ATAM, SAAM y ALMA. La propuesta que utilizan en el trabajo funciona para medir la modificabilidad que presenta la arquitectura compuesta con base en el número de componentes, las conexiones directas e indirectas entre estos. Este proceso se adecuará de acuerdo con las necesidades de esta investigación para poder utilizar las fórmulas propuestas para la evaluación de la modificabilidad existente en la arquitectura.

5.1. Construcción del caso de estudio

En este apartado, se describe el proceso que se realizó para el desarrollo de un caso de estudio en las plataformas FIWARE y DeviceHive, tomando en cuenta que ambas plataformas son muy similares en su funcionamiento. Solo se representará cuando sea necesario mostrar las diferencias entre estos refiriéndose al experimento a realizar.

El envío de los datos de posicionamiento es realizado enviado datos en formato JSON de dos formas distintas. La primera es desde el mismo dispositivo móvil, en el cual se representará la función de la capa de Servicio de posición enviando datos a la plataforma IoT en cuestión, la segunda manera es mediante un programa que simula el envío de datos a la plataforma IoT correspondiente de múltiples dispositivos de forma continua en un patrón repetitivo.

A partir de estos datos se puede procesar la información dependiendo de la plataforma, para la plataforma de FIWARE el flujo de datos es el siguiente: Orion Context Broker recibirá las entidades de datos que son enviados por la capa anterior, a su vez este notificará al servicio de QuantumLeap que se encargará de transformar el modelo de datos y transformarlo en registros para una base de datos alojada en CrateDB donde se almacenarán de acuerdo con el tiempo que fueron recibidos. Para la plataforma de DeviceHive el flujo de las entidades es el siguiente: Los datos son recibidos por el servicio de Frontend de DeviceHive y este transmite la información al Broker Kafka mandando la información necesaria al servicio de Backend que procederá a almacenar la información de las entidades

en el servicio de cache (Hazelcast) y si es necesario después la información es transferida a la base de datos de PostgreSQL para el almacenamiento persistente.

Una vez que los datos ya se encuentran disponibles para consumo en cualquiera de las plataformas IoT, la capa de aplicación donde se encuentran los servicios web podrá consumir los datos de los dispositivos móviles que se encuentren activos, los datos son transformados y almacenados en variables para que puedan ser desplegados de forma visual en la interfaz web del mapa del edificio, y a su vez se muestren notificaciones si algún dispositivo sale del área de monitoreo.

5.1.1. Escenario de pruebas

El escenario de pruebas está delimitado a una zona en una casa habitación con un área aproximada de 85 m² y este se conforma de la habitación principal, el área de closet, cuarto de baño, sala de estar y terraza como se muestra en la Figura 21.

Figura 21 Escenario de pruebas

Para finalidades prácticas de este trabajo se simularán múltiples personas de forma estática y dinámica dentro del mapa, representados mediante una imagen de una persona en la posición que indiquen las coordenadas que reciben las plataformas IoT.

5.2. Caso de estudio enfocado a FIWARE

Para poder realizar una evaluación a la arquitectura con dos plataformas distintas es necesario dividir la arquitectura en sistemas separados uno del otro, en este caso utilizando solo componentes de FIWARE para su funcionamiento. La Figura 22 muestra la simplificación de la arquitectura que se muestra en la Figura 10, donde solo se utilizan los componentes presentes en FIWARE para el manejo de datos de posición de los dispositivos simulados.

Se mantienen los componentes de las capas externas (Interfaz y Posicionamiento), y en la capa intermedia se utilizan los componentes necesarios para la comunicación y almacenamiento de datos de posicionamiento que se envían y reciben de las capas ya mencionadas.

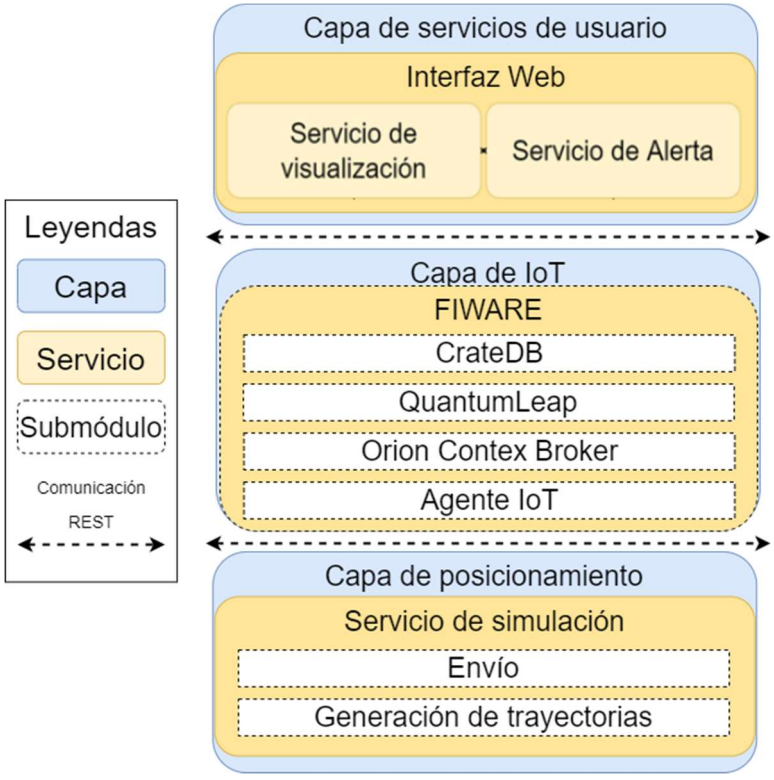

Figura 22 Arquitectura para caso de estudio de FIWARE

El funcionamiento general de FIWARE para este caso es el siguiente: Primero los datos son generados por la capa de posicionamiento (Dispositivo móvil o el servicio de

simulación), después son enviados al Orion Context Broker de FIWARE utilizando el agente correspondiente para el protocolo REST, y éste notificará a QuantumLeap para el almacenamiento y consulta de los modelos de datos que se encuentren disponibles para consulta. Por último, los datos pueden ser consultados por los servicios web mediante consultas REST de los dispositivos específicos para el usuario en cuestión.

5.3. Caso de estudio enfocado a DeviceHive

De la misma manera, para este caso de estudio, se simplifica la arquitectura mostrada en la Figura 10, donde solo se utilizan los componentes de la plataforma DeviceHive como se muestra en la Figura 23.

Como se puede apreciar, de igual forma que el caso de estudio para la plataforma de FIWARE, se mantienen igual las capas exteriores (Interfaz y Posicionamiento), mientras que la capa intermedia utiliza los componentes necesarios para mantener la comunicación entre capas y el almacenamiento de los esquemas de datos de posición que se están enviando y recibiendo de forma constante durante la ejecución.

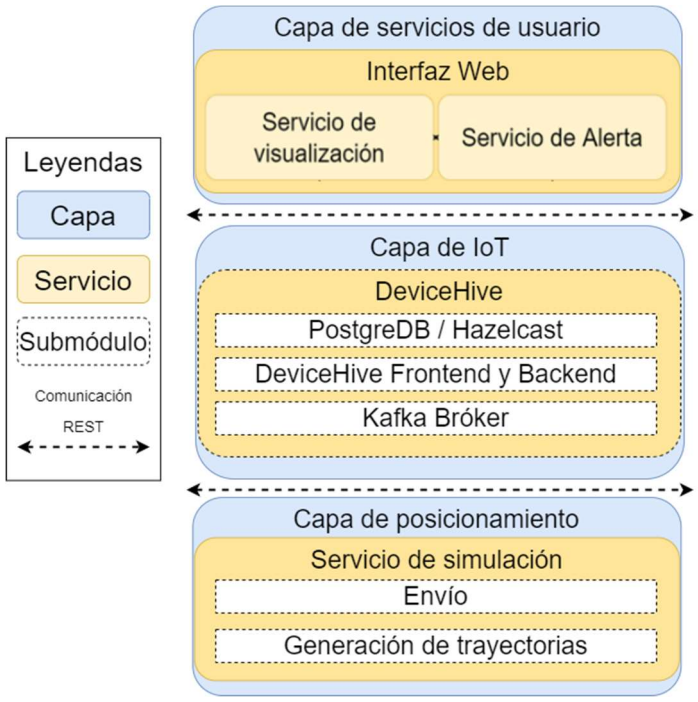

Figura 23 Arquitectura para caso de estudio de DeviceHive

El funcionamiento general de DeviceHive para este caso es el siguiente: Primero los datos son generados por la capa de Posicionamiento (Dispositivo móvil o el servicio de simulación), los datos son enviados mediante protocolo REST al servicio de Frontend de DeviceHive para que mediante el bróker Kafka notifique al servicio de Backend para el almacenamiento de los datos ya sea en Hazelcast o Postgres, para que puedan ser accedidos mediante peticiones REST en cualquier momento. Por último, los datos pueden ser consultados por los servicios web mediante consultas REST de los dispositivos específicos para el usuario en cuestión.

5.4. Caso de estudio con ambas plataformas

Una vez evaluados los componentes de las plataformas IoT de forma independiente, es pertinente evaluar el comportamiento de la plataforma en términos de modificabilidad cuando ambas plataformas trabajan de forma paralela para proporcionar las mismas funcionalidades necesarias para el funcionamiento del sistema.

La Figura 24 presenta el diagrama de la arquitectura donde se incorporan los elementos necesarios de ambas plataformas y se configuran los servicios para trabajar de forma simultánea para enviar y recibir datos a ambas plataformas.

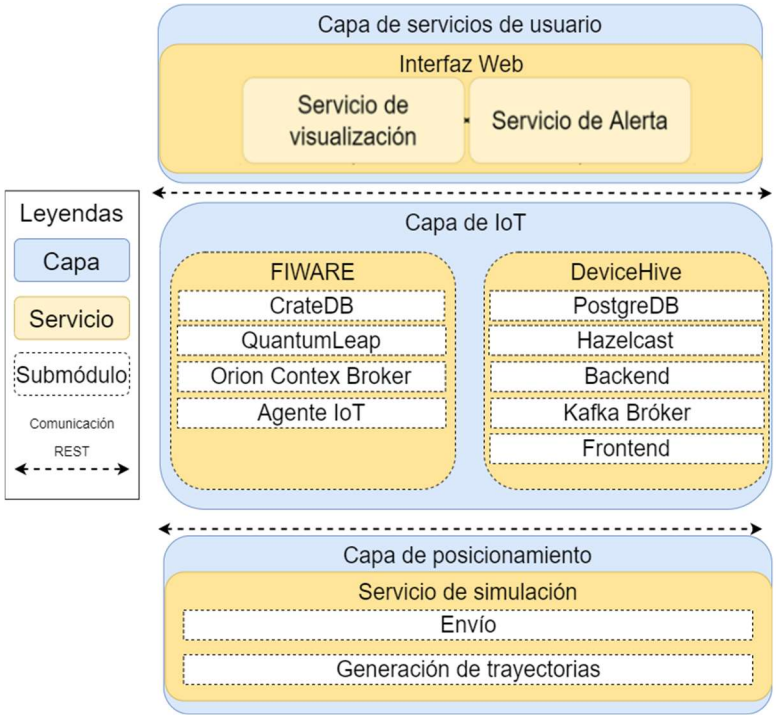

Figura 24 Arquitectura para caso de estudio simultáneo

Como se puede apreciar, la estructura y funcionamiento se mantiene prácticamente igual que en los anteriores casos de estudio, las capas de posicionamiento y de servicios se adaptan para el envío y recepción de datos a ambas plataformas, mientras que la capa intermedia se permite el uso de ambas plataformas, sin relacionarse, pero trabajando de forma paralela para ofrecer los servicios correspondientes a las capas inferior y superior en la arquitectura.

Capítulo 6

Pruebas y evaluación de resultados

6. Plan de pruebas

La funcionalidad de un sistema se define por su estructura y cómo funciona. Esta estructura se puede representar en un conjunto de reglas que describen cómo se debe construir y operar el sistema. El impacto de los cambios en estas especificaciones es difícil de determinar porque se requiere el conocimiento de cómo funcionan los sistemas. Para determinar que tan bien se ha diseñado, primero se debe comprender las subestructuras y después determinar qué cambios afectarán a dicha estructura.

Para evaluar la modificabilidad en la arquitectura se usó un enfoque cuantitativo para así dar una prueba inequívoca de que se cumple con altos niveles en la modificabilidad. El enfoque propuesto se basa en la suposición de cuando se diseñan los sistemas, estos por lo general se diseñan para solventar tareas específicas, esto debido a que los patrones arquitectónicos buscan solucionar atributos de calidad específicos a cambio de descuidar otros no tan importantes.

Utilizando las técnicas implementadas en [36] donde se evalúa la modificabilidad de forma matemática en una arquitectura compuesta, se puede realizar un análisis que permita medir la modificabilidad en la arquitectura propuesta. El objetivo de este análisis es determinar cuánto esfuerzo tomaría para modificar los elementos del sistema sin alterar el resto de los componentes. Esto nos permite identificar áreas en las que es probable o improbable que ocurran cambios, y, por lo tanto, comenzar a comprender la arquitectura a niveles de modificabilidad y desarrollo.

Para lograr esto es necesario realizar una adaptación al trabajo ya mencionado [36], donde realizan una evaluación a los filtros o procesos que se realizan en el sistema, es decir por los componentes y procesos por los que pasa cada el flujo de datos en el sistema, por esto, se toma en cuenta como equivalente el componente de la arquitectura por el cual pasa información desde que se origina hasta que es consumida por el usuario, el número de enlaces directos e indirectos se mantiene de la misma manera, ya que solo nos indica en este caso, la conexión con otro componente para los directos, en caso de los indirectos nos indica el número de enlaces sin conexiones directas. Por lo tanto, las fórmulas que utilizaron en el trabajo [30] son válidas para evaluar este trabajo.

6.1. Evaluación de la arquitectura

El objetivo de la evaluación de la arquitectura es valorar la modificabilidad que presentan los casos de estudio, evaluándolos por separado y en conjunto. La puntuación de la modificabilidad se determina con cálculos que tienen base en el acoplamiento de los componentes y la complejidad de éstos, los resultados de las evaluaciones serán comparados con las métricas de modificabilidad presentados en el trabajo [36] que se muestra en la Tabla 5 obtenida del mismo.

Tabla 5. Métrica de Modificabilidad

Factor de acoplamiento	Umbral de modificabilidad	Evaluación de la arquitectura
[0]	Muy alto	Excelente
(0-0.5)	>2	Bueno
[0.5]	2	Altamente aceptable
(0.5-1)	1<m<2	Aceptable
[1]	1	No aceptable

El formato de evaluación consistirá en el siguiente formato, donde se describirán las variables *M*, para indicar el tipo de ejecución (1 para secuencial, 2 para paralelo, etc.), *N*, para indicar el número de componentes que interactúan en el sistema, *p*, para indicar el número de relaciones directas y describirlas en formato (C1C2, indicando una relación entre componente 1 y componente 2) en una lista simple, *q*, para indicar el número de relaciones no directas y describirlas en el mismo formato en una lista simple. Una vez descritos todos los valores proceder a utilizar las ecuaciones 1 y 2 [36] para calcular el acoplamiento y la modificabilidad respectivamente.

1) $Acoplamiento = \frac{p}{(p+q)}$

2) $Modificabilidad = M \times \frac{1}{Acoplamiento}$

Posteriormente de obtener los resultados comparar los resultados con la métrica de modificabilidad presentada en la Tabla 5.

6.1.1. Evaluación de la arquitectura usando FIWARE

A continuación, se listan los componentes utilizados, las variables y los resultados de la evaluación del caso de estudio de la arquitectura utilizando solo componentes de la plataforma FIWARE.

Componentes:

C1.- Posicionamiento

C2.- Agente IoT para JSON

C3.- Orion Context Broker

C4.- QuantumLeap

C5.- CrateDB

C6.- Servicios web

Variables:

Tabla 6 Variables FIWARE

Variable	Valor	Descripción
M	1	Ejecución secuencial
N	6	Número de componentes
p	5	C1C2, C2C3, C3C4, C4C5, C5C6
q	10	C1C3, C1C4, C1C5, C1C6, C2C4, C2C5, C2C6, C3C5, C3C6, C4C6

Evaluación:

Evaluando las ecuaciones 1 y 2 con las variables se obtienen los siguientes resultados:

$$1)\quad Acoplamiento = \frac{p}{(p+q)} = \frac{5}{5+1} = \frac{1}{3} = 0.\underline{3}$$

$$2)\quad Modificabilidad = M \times \frac{1}{Acoplamiento} = 1 \times \frac{1}{\frac{1}{3}} = 3$$

De acuerdo con los resultados obtenidos y comparando los resultados con la Tabla 5, un factor de acoplamiento de 0.333 y una modificabilidad de 3 se considera como una arquitectura con un Buen nivel de modificabilidad, lo cual es esperado al trabajar con múltiples componentes que se pueden comunicar fácilmente entre sí.

6.1.2. Evaluación de la arquitectura usando DeviceHive

A continuación, se listan los componentes utilizados, las variables y los resultados de la evaluación del caso de estudio de la arquitectura utilizando solo componentes de la plataforma DeviceHive.

Componentes:

C1.- Posicionamiento

C2.- Servicio de Frontend DH

C3.- Kafka Broker

C4.- Servicio de Backend DH

C5.- Hazelcast

C6.- Postgres

C7.- Servicios web

Variables:

Tabla 7 Variables DeviceHive

Variable	Valor	Descripción
M	1	Ejecución secuencial
N	7	Número de componentes
p	6	C1C2, C2C3, C3C4, C4C5, C5C6, C6C7
q	15	C1C3, C1C4, C1C5, C1C6, C1C7, C2C4, C2C5, C2C6, C2C7, C3C5, C3C6, C3C7, C4C6, C4C7, C5C7

Evaluación:

Evaluando las ecuaciones 1 y 2 con las variables se obtienen los siguientes resultados:

$$1)\quad Acoplamiento = \frac{p}{(p+q)} = \frac{6}{6+15} = \frac{6}{21} = 0.2857$$

$$2)\quad Modificabilidad = M \times \frac{1}{Acoplamiento} = 1 \times \frac{1}{\frac{6}{21}} = 3.5$$

De acuerdo con los resultados obtenidos y comparando los resultados con la Tabla 5, un factor de acoplamiento de 0.2857 y una modificabilidad de 3.5 se considera como una arquitectura con un Buen nivel de modificabilidad, lo cual es esperado al trabajar con múltiples componentes que se pueden comunicar fácilmente entre sí.

6.1.3. Evaluación de la arquitectura usando FIWARE y DeviceHive

A continuación, se listan los componentes utilizados, las variables y los resultados de la evaluación del caso de estudio de la arquitectura utilizando de forma simultánea los componentes de las plataformas FIWARE y DeviceHive

Componentes:

C1.- Posicionamiento
C2.- Servicio de Frontend DH
C3.- Kafka Broker
C4.- Servicio de Backend DH
C5.- Hazelcast
C6.- PostgreSQL
C7.- Servicios web
C8.- Posicionamiento
C9.- Agente IoT para JSON
C10.- Orion Context Broker
C11.- QuantumLeap
C12.- CrateDB
C13.- Servicios web

Variables:

Tabla 8 Variables del caso de estudio

Variable	Valor	Descripción
M	2	Ejecución paralela
N	13	Número de componentes
p	11	C1C2, C2C3, C3C4, C4C5, C5C6, C6C7 C8C9, C9C10, C10C11, C11C12, C12C13
q	67	C1C3, C1C4, C1C5, C1C6, C1C7, C1C8, C1C9, C1C10, C1C11, C1C12, C1C13, C2C4, C2C5, C2C6, C2C7, C2C8, C2C9, C2C10, C2C11, C2C12, C2C13, C3C5, C3C6, C3C7, C3C8, C3C9, C3C10, C3C11, C3C12, C3C13, C4C6, C4C7, C4C8, C4C9, C4C10, C4C11, C4C12, C4C13, C5C7, C5C8, C5C9, C5C10, C5C11, C5C12, C5C13, C6C8, C6C9, C6C10, C6C11, C6C12, C6C13, C7C8, C7C9, C7C10, C7C11, C7C12, C7C13, C8C10, C8C11, C8C12, C8C13, C9C11, C9C12, C9C13, C10C12, C10C13, C11C13

Evaluación:

Evaluando las ecuaciones 1 y 2 con las variables se obtienen los siguientes resultados:

1) $Acoplamiento = \frac{p}{(p+q)} = \frac{11}{11+} = \frac{11}{78} = 0.1410$

2) $Modificabilidad = M \times \frac{1}{Acoplamiento} = 2 \times \frac{1}{\frac{11}{78}} = 14.18$

De acuerdo con los resultados obtenidos y comparando los resultados con la Tabla 5, un factor de acoplamiento de 0.1410 y una modificabilidad de 14.18 se considera como una arquitectura con un Buen nivel de modificabilidad, si bien la modificabilidad creció de manera notable al utilizar componentes en paralelo, el factor de acoplamiento aún no es lo suficientemente bajo (0) para considerarse una arquitectura con un nivel excelente de modificabilidad, pero aun así trabajar con múltiples componentes ayuda a bajar el nivel de acoplamiento y por lo tanto, incrementar la modificabilidad del sistema.

6.2. Comparación de resultados

Conforme a los resultados en las tres evaluaciones se puede observar que el uso de la arquitectura propuesta ofrece un nivel de modificabilidad aceptable, esto debido al uso de múltiples componentes de las plataformas IoT, así como la planeación para el desarrollo de componentes modulares y de fácil intercambio que permite la arquitectura en cada capa.

La Tabla 9 se muestran los resultados de las evaluaciones realizadas a la arquitectura, como se puede apreciar los resultados de la tercera evaluación, al utilizar ambas plataformas de forma paralela el valor de acoplamiento decrece, lo cual es favorable si se busca aumentar el nivel de modificabilidad.

Tabla 9 Resumen de resultados

No.	*M*	*N*	*p*	*q*	Acoplamiento	Modificabilidad	Evaluación
FIWARE	1	6	5	10	0.3333	3	Buena
DeviceHive	1	7	6	15	0.2857	3.5	Buena
FW/DH	2	13	11	67	0.1410	14.18	Buena

Capítulo 7

Conclusiones y Trabajo Futuro

7. Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones del desarrollo de la investigación, así como recomendaciones que pueden derivar en investigaciones futuras.

7.1. Conclusiones

Esta tesis tuvo como objetivo desarrollar una arquitectura orientada a servicios enfocada en la localización de personas en espacios cerrados, capaz de soportar crear, implementar, modificar y eliminar diversos servicios de manera ágil.

Para el diseño de la arquitectura se optó por un diseño compuesto utilizando el patrón en capas como base. Este patrón tiene como característica principal actualizar con adecuada facilidad componentes, brindando flexibilidad y modificabilidad a los diversos elementos del sistema. Además, se implementaron mecanismos de comunicación estandarizados entre las capas para ofrecer una funcionalidad similar a la arquitectura orientada a servicios, lo cual a su vez otorga ventajas asociadas de flexibilidad y agilidad para el desarrollo y mantenimiento del sistema. Consideramos que el patrón en capas proporciona una estructura estable para el desarrollo inicial de proyectos, ya que permite segmentar el sistema para futuras modificaciones, así como la integración de servicios de una manera más simple. La modularidad que brinda el patrón de capas permite la modificabilidad y portabilidad de los componentes desarrollados, y a su vez que estos componentes puedan ser distribuidos en distintos servidores para así reducir la carga de procesamiento y almacenamiento de datos.

La arquitectura se implementó en un entorno Web para facilitar la visualización desde cualquier dispositivo que cuente con un navegador compatible con las tecnologías utilizadas. Se utilizó PHP como lenguaje principal y el estándar REST para la comunicación entre las capas utilizando mensajes JSON con información estructurada. La aplicación presenta dos funcionalidades: 1) despliega un mapa del edificio con los datos de ubicación de personas que se encuentran dentro del edificio, y 2) despliega alertas en formato de texto en el caso que algún usuario salga del área de monitoreo. Los componentes tanto de FIWARE como DeviceHive fueron alojados en distintos servidores con la finalidad de mostrar la escalabilidad de los componentes, y el resto de los servicios en un servidor local para facilitar el desarrollo e implementación del sistema.

Para evaluar la modificabilidad en la arquitectura se utilizó el método cuantitativo propuesto por [36]. Este método permite determinar el esfuerzo que toma modificar los elementos del sistema sin alterar el resto de los componentes. Se proponen métricas específicas para medir la modificabilidad de una arquitectura de software obteniendo resultados satisfactorios.

De acuerdo con el método [36], se utilizaron fórmulas que permiten medir el acoplamiento y el umbral de modificabilidad de la arquitectura considerando los siguientes factores: Tipo de ejecución de los procesos (secuencial o en paralelo), número total de componentes en interacción, número de relaciones directas de componente a componente. Se evaluaron los 3 casos de estudio, utilizando FIWARE, utilizando DeviceHive y utilizando de manera simultánea FIWARE y DeviceHive. De acuerdo con este método se tienen 5 valores de desempeño: Excelente, Bueno, Altamente aceptable, Aceptable y No aceptable. Para cada uno de los 3 casos de estudio se obtuvo un desempeño Bueno.

7.2. Trabajos futuros

Con el crecimiento de la población y el crecimiento de ciudades con poblaciones más densas, la implementación de servicios de monitoreo en espacios cerrados será relevante en los años siguientes. Los trabajos futuros que se proponen para ampliar la investigación son:

- Investigar otros métodos de evaluación de la modificabilidad en arquitecturas para obtener resultados validados de forma múltiple, con la finalidad de asegurar que se cuenta con una arquitectura ágil para implementar servicios de posicionamiento.
- Realizar pruebas de funcionamiento en entornos reales con múltiples personas, señales y algoritmos, para analizar rendimiento, usabilidad, escalabilidad y despliegue según se requiera en la investigación correspondiente.
- Consultar plataformas de código libre (diferentes a FIWARE y DeviceHive) que se puedan utilizar con la arquitectura planteada, para no limitar el trabajo a solo dos plataformas.
- Realizar experimentos de implementación ágil donde se utilicen programadores de cualidades similares para desarrollar e implementar un servicio de posicionamiento definido, donde se evaluará factores relacionados con el experimento de forma

cuantitativa (líneas de código, tiempo de desarrollo, complejidad algorítmica O) y cualitativa mediante formularios para los programadores describan su experiencia para desarrollar el servicio, su nivel de adaptación a la arquitectura, entre otros factores relevantes.

- Agregar información adicional a los modelos de datos con la finalidad de incluir nuevas funcionalidades y servicios como la detección de caídas [52], o datos de otros dispositivos vestibles que detecten la ansiedad del usuario [53] o la actividad física realizada [54].
- Considerar el uso de domótica en combinación con preferencias del usuario para facilitar aspectos de la vida diaria para generar un entorno más inteligente y al servicio del usuario.

Referencias

[1] N. E. Klepeis *et al.*, "The National Human Activity Pattern Survey (NHAPS): A resource for assessing exposure to environmental pollutants," *J Expo Anal Environ Epidemiol*, vol. 11, no. 3, pp. 231–252, 2001, doi: 10.1038/sj.jea.7500165.

[2] D. Zhang, F. Xia, Z. Yang, L. Yao, and W. Zhao, "Localization technologies for indoor human tracking," *2010 5th International Conference on Future Information Technology, FutureTech 2010 - Proceedings*, no. 60903153, 2010, doi: 10.1109/FUTURETECH.2010.5482731.

[3] E. Ferro and F. Potortì, "Bluetooth and Wi-Fi wireless protocols: A survey and a comparison," *IEEE Wirel Commun*, vol. 12, no. 1, pp. 12–26, 2005, doi: 10.1109/MWC.2005.1404569.

[4] "Arquitectura orientada a servicios - Documentación de IBM." https://www.ibm.com/docs/es/bpm/8.6.0?topic=designer-service-oriented-architecture (accessed May 16, 2022).

[5] "Las ciudades seguirán creciendo, sobre todo en los países en desarrollo | ONU DAES | Naciones Unidas Departamento de Asuntos Económicos y Sociales." https://www.un.org/development/desa/es/news/population/2018-world-urbanization-prospects.html (accessed May 16, 2022).

[6] V. Gutiérrez *et al.*, "Co-creating the cities of the future," *Sensors (Switzerland)*, vol. 16, no. 11, pp. 1–27, 2016, doi: 10.3390/s16111971.

[7] F. Gu *et al.*, "Indoor localization improved by spatial context - A survey," *ACM Comput Surv*, vol. 52, no. 3, 2019, doi: 10.1145/3322241.

[8] P. Davidson and R. Piché, "A Survey of Selected Indoor Positioning Methods for Smartphones," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 2, pp. 1347–1370, 2017, doi: 10.1109/COMST.2016.2637663.

[9] S. He and K. G. Shin, "Geomagnetism for Smartphone-Based Indoor Localization," *ACM Comput Surv*, vol. 50, no. 6, pp. 1–37, 2018, doi: 10.1145/3139222.

[10] A. Tahat, G. Kaddoum, S. Yousefi, S. Valaee, and F. Gagnon, "A Look at the Recent Wireless Positioning Techniques with a Focus on Algorithms for Moving Receivers," *IEEE Access*, vol. 4, no. c, pp. 6652–6680, 2016, doi: 10.1109/ACCESS.2016.2606486.

[11] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice , Second Edition*. 2003.

[12] G. Choudhary and A. K. Jain, "Internet of Things: A survey on architecture, technologies, protocols and challenges," *2016 International Conference on Recent Advances and Innovations in Engineering, ICRAIE 2016*, 2016, doi: 10.1109/ICRAIE.2016.7939537.

[13] K. Avila, P. Sanmartin, D. Jabba, and M. Jimeno, "Applications based on service-oriented architecture (SOA) in the field of home healthcare," *Sensors (Switzerland)*, vol. 17, no. 8, 2017, doi: 10.3390/s17081703.

[14] "About FIWARE - FIWARE." https://www.fiware.org/about-us/ (accessed May 29, 2022).

[15] "Developers Catalogue - FIWARE." https://www.fiware.org/catalogue/ (accessed May 29, 2022).

[16] “DeviceHive - Open Source IoT Data Platform with the wide range of integration options.” https://devicehive.com/#open+source (accessed May 29, 2022).
[17] “Three Steps To IoT.” https://docs.devicehive.com/docs (accessed May 29, 2022).
[18] S. Hill, “Scalable IoT Platforms,” p. 141, 2019, [Online]. Available: http://elib.uni-stuttgart.de/handle/11682/10483
[19] E. Newcomer, *Understanding SOA with Web services*. Pearson Education India, 2005.
[20] “What is an API? (Application Programming Interface) | MuleSoft.” https://www.mulesoft.com/resources/api/what-is-an-api (accessed May 17, 2022).
[21] M. Mancasi, R. Vatu, O. Ceaki, R. Porumb, and G. Seritan, “Evolution of smart buildings. A Romanian case,” *Proceedings of the Universities Power Engineering Conference*, vol. 2015-Novem, pp. 1–4, 2015, doi: 10.1109/UPEC.2015.7339802.
[22] Q. Lê, H. B. Nguyen, and T. Barnett, “Smart Homes for Older People: Positive Aging in a Digital World,” *Future Internet*, vol. 4, no. 2, pp. 607–617, 2012, doi: 10.3390/fi4020607.
[23] J. al Dakheel, C. del Pero, N. Aste, and F. Leonforte, “Smart buildings features and key performance indicators: A review,” *Sustain Cities Soc*, vol. 61, p. 102328, 2020, doi: 10.1016/j.scs.2020.102328.
[24] J. Fombona Cadavieco and E. Vázquez-Cano, “Posibilidades de utilización de la geolocalización y realidad aumentada en el ámbito educativo,” *Educacion XX1*, vol. 20, no. 2, pp. 319–342, 2017, doi: 10.5944/educXX1.10852.
[25] D. Katherine and F. Taipe, “Sistema de Localización Indoor y Outdoor para un Mini Vehículo Aéreo Autónomo No Tripulado utilizando Módulos Wi-Fi,” 2017.
[26] M. Er Rida, F. Liu, Y. Jadi, A. A. A. Algawhari, and A. Askourih, “Indoor location position based on bluetooth signal strength,” *Proceedings - 2015 2nd International Conference on Information Science and Control Engineering, ICISCE 2015*, pp. 769–773, 2015, doi: 10.1109/ICISCE.2015.177.
[27] “Discover Wi-Fi | Wi-Fi Alliance.” https://www.wi-fi.org/discover-wi-fi (accessed May 17, 2022).
[28] “Bluetooth Technology Overview | Bluetooth® Technology Website.” https://www.bluetooth.com/learn-about-bluetooth/tech-overview/ (accessed May 17, 2022).
[29] A. Martínez, L. A. López, H. Estrada, M. Mejía, and J. Ortiz, “Navegación en interiores utilizando la tecnología beacons,” in *Research in Computing Science*, 2019.
[30] D. Cervantes, A. Martínez, J. Mosiño, and J. Ortiz, “Localización de dispositivos móviles en espacios cerrados utilizando la tecnología de beacons,” *Jornada de Ciencia y Tecnología Aplicada*, vol. 2, 2019, [Online]. Available: https://www.cenidet.edu.mx/archivos/electronica/revista%5C JCyTA/Revista%5C JCyTA%5C Vol.%5C 2.%5C Abril%5C de%5C 2019.pdf
[31] M. A. Yris, “API para servicios de localización en interiores basada en tecnología Wi-Fi, Bluetooth, RFID y QRCode,” Centro Nacional de Investigación y Desarrollo Tecnológico, Interior Internado Palmira S/N, Col. Palmira, C.P. 62490, Cuernavaca, Morelos., 2012.
[32] B. R. Ramírez, “Sistema de Información Aumentada para navegación Indoor-Outdoor Basado en Modelos Semánticos Organizacionales,” Tecnológico Nacional de México / CENIDET, Interior Internado Palmira S/N, Col. Palmira, C.P. 62490, Cuernavaca, Morelos., 2015.

[33] I. Arjona, “Servicios de localización conscientes del contexto aplicando perfiles de movilidad y tecnologías de localización heterogéneas,” Centro Nacional de Investigación y Desarrollo Tecnológico, Interior Internado Palmira S/N, Col. Palmira, C.P. 62490, Cuernavaca, Morelos., 2009.

[34] R. Infante, J. Ortiz, A. Martínez, and S. García, “Geolocalización y estudio de la movilidad de personas en espacios cerrados a partir de emisores WiFi y beacons,” *Jornada de Ciencia y Tecnología Aplicada*, vol. 2, p. 99, Apr. 2019, [Online]. Available: https://www.cenidet.edu.mx/archivos/electronica/revista%5C JCyTA/Revista%5C JCyTA%5C Vol.%5C 2.%5C Abril%5C de%5C 2019.pdf

[35] R. Alshinina and K. Elleithy, “Performance and challenges of Service-Oriented architecture for wireless sensor networks,” *Sensors (Switzerland)*, vol. 17, no. 3, 2017, doi: 10.3390/s17030536.

[36] M. M. Philip, N. Singhal, R. Ravi, and B. Vijayakumar, “A quantitative approach to analyze modifiability in software architectural design of agile application systems,” *Information Technology and Control*, vol. 49, no. 2, pp. 249–259, Sep. 2020, doi: 10.5755/j01.itc.49.2.22893.

[37] L. Zhang, H. Yuan, S. H. Chang, and A. Lam, “Research on the overall architecture of Internet of Things middleware for intelligent industrial parks,” *The International Journal of Advanced Manufacturing Technology 2019 107:3*, vol. 107, no. 3, pp. 1081–1089, Sep. 2019, doi: 10.1007/S00170-019-04310-Z.

[38] P. Pierleoni, R. Concetti, A. Belli, and L. Palma, “Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison,” *IEEE Access*, vol. 8, pp. 5455–5470, 2020, doi: 10.1109/ACCESS.2019.2961511.

[39] S. P. Gochhayat *et al.*, “LISA: Lightweight context-aware IoT service architecture,” *J Clean Prod*, vol. 212, pp. 1345–1356, 2019, doi: 10.1016/j.jclepro.2018.12.096.

[40] J. Ortiz-Hernandez, J. A. Miguel-Ruiz, M. Erazo-Valadez, L. Torres-Restrepo, A. Martinez-Rebollar, and M. Mejia-Lavalle, “A Lightweight Framework for IoT Smart Solutions,” in *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, Dec. 2020, pp. 1149–1154. doi: 10.1109/CSCI51800.2020.00215.

[41] S. Sadowski and P. Spachos, “Comparison of RSSI-Based Indoor Localization for Smart Buildings with Internet of Things,” *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON 2018*, pp. 24–29, Jan. 2019, doi: 10.1109/IEMCON.2018.8614863.

[42] M. H. Valipour, B. Amirzafari, K. N. Maleki, and N. Daneshpour, “A brief survey of software architecture concepts and service oriented architecture,” *Proceedings - 2009 2nd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2009*, pp. 34–38, 2009, doi: 10.1109/ICCSIT.2009.5235004.

[43] “Reference Model for Service Oriented Architecture v1.0.” https://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html (accessed May 22, 2022).

[44] A. Shashwat and D. Kumar, “A service identification model for service oriented architecture,” *3rd IEEE International Conference on* , Jul. 2017, doi: 10.1109/CIACT.2017.7977299.

[45] “Situm Rest API.” https://developers.situm.com/pages/rest/openapi/ (accessed May 22, 2022).

[46] “Mapwize | Support Center.” https://docs.mapwize.io/ (accessed May 22, 2022).

[47] “Indoora – Indoor Positioning Systems – Services.” https://www.indoora.com/services/ (accessed May 22, 2022).
[48] Navisens, “How Does It Work.” 2020.
[49] Litum, “How Our Location Tracking System Works.” 2020.
[50] “¿Qué es la arquitectura orientada a los servicios?” https://www.redhat.com/es/topics/cloud-native-apps/what-is-service-oriented-architecture (accessed Jul. 10, 2022).
[51] A. Abella-Garcia, “DataModel.Device.” https://github.com/smart-data-models/dataModel.Device/blob/master/Device/doc/spec_ES.md (accessed Jan. 22, 2023).
[52] M. Erazo Valadez, J. Ortiz Hernández, and M. Y. Hernández Pérez, “Implementación del método de umbrales para la detección de caídas utilizando sensores vestibles,” Centro Nacional de Investigación y Desarrollo Tecnológico, Cuernavaca, 2022. Accessed: Nov. 27, 2022. [Online]. Available: https://rinacional.tecnm.mx/jspui/handle/TecNM/4160
[53] P. A. Cuevas Chávez and A. Martínez Rebollar, “Detección Automática de Ansiedad a Través del Monitoreo de Sensores Vestibles,” Centro Nacional de Investigación y Desarrollo Tecnológico, Cuernavaca, 2020. Accessed: Nov. 27, 2022. [Online]. Available: https://rinacional.tecnm.mx/jspui/handle/TecNM/2917
[54] J. A. Miguel Ruiz and J. Ortiz Hernández, “Medición del gasto energético en adultos utilizando sensores vestibles,” Centro Nacional de Investigación y Desarrollo Tecnológico, Cuernavaca, 2022. Accessed: Nov. 27, 2022. [Online]. Available: https://rinacional.tecnm.mx/jspui/handle/TecNM/4158

Anexo A
Productos académicos

Los logros obtenidos a partir del desarrollo de esta investigación son listados a continuación:

1. Artículo de conferencia *Análisis comparativo de arquitecturas para Internet of Things.* Presentado en la Jornada de Ciencia y Tecnología Aplicada Vol. 3, Núm. 1m Enero – Junio 2020.
1. Artículo de conferencia *Implementación Ágil de Servicios para Aplicaciones de Internet de las Cosas.* Presentado en la Jornada de Ciencia y Tecnología Aplicada Vol. 3, Núm. 2, Julio – Diciembre 2020.

Análisis comparativo de arquitecturas para Internet of Things

Erick Infante-Covarrubias * Javier Ortiz-Hernández *

* *Tecnológico Nacional de México/CENIDET Cuernavaca, Morelos, México (e-mail: {erick.infante19ca, ortiz}@cenidet.edu.mx).*

Resumen:
El desarrollo de la infraestructura y las tecnologías que apoyan la accesibilidad, la vida activa y asistida son importantes, especialmente en el contexto del desarrollo de ciudades inteligentes con el uso del internet de las cosas (IoT, Internet of Things). El objetivo de este trabajo es presentar una comparación de tres de las arquitecturas más utilizadas en sistemas enfocados a IoT, destacando las ventajas y desventajas de cada una de ellas.

Palabras Clave: Internet of Things, IoT, System architecture, Service oriented, Sensor networks, Cloud, Blockchain

1. INTRODUCCIÓN

Las tecnologías de la información son factor clave en la mejora de la calidad de vida y un apoyo fundamental para las actividades diarias de las personas y las organizaciones, ya que permite incrementar la productividad, sistematizar procesos, mejorar la comunicación, etc. Para ello, se busca el diseño y desarrollo de herramientas que le permiten a las personas y objetos estar conectados en tiempo real, intercambiar información y auxiliar diferentes tareas cotidianas. El desarrollo de herramientas de esta naturaleza ha dado lugar a las diferentes aplicaciones del IoT, tales como automatización en casas inteligentes, seguimiento en espacios cerrados, conectividad entre dispositivos y aplicaciones móviles.

Una arquitectura de servicios puede integrar las mejores alternativas tecnológicas para el desarrollo de aplicaciones de IoT, permitiendo superar los desafíos que presentan este tipo de proyectos (Li & Zhang, 2017).

El objetivo de una arquitectura de servicios es separar la lógica de integración del negocio de la implementación. Para lograr esto, se necesita la creación de componentes que contengan la implementación de los servicios individuales necesarios para los procesos de negocio (IBM, 2020b). Esta solución permite explotar de una manera eficaz y eficiente la creación de nuevos servicios para el IoT, de manera más ágil, en un menor tiempo y en consecuencia con un menor costo.

El objetivo de este trabajo es llevar a cabo un estudio comparativo de las principales características, ventajas y desventajas de algunas de las arquitecturas disponibles para el desarrollo de aplicaciones de IoT. El artículo está estructurado de la siguiente manera: la Sección 2 describe los desafíos actuales a los que se enfrenta el desarrollo de sistemas de IoT; la Sección 3 describe los criterios con los cuales se analizaron las arquitecturas; la Sección 4 describe las características principales de las arquitecturas; la Sección 5 presenta los resultados del estudio comparativo entre las arquitecturas seleccionadas; y la Sección 6 presenta las conclusiones de este trabajo.

2. DESAFÍOS DEL IoT

En esta sección se presenta una breve revisión de los desafíos clave a los que se enfrenta la tecnología para el desarrollo de soluciones basadas en IoT y que se pueden sintetizar en los siguientes: confiabilidad, movilidad, disponibilidad, escalabilidad, rendimiento y problemas de interoperabilidad. (Al-Fuqaha et al., 2015; Sisinni et al., 2018; Yaqoob et al., 2017).

- **Confiabilidad** La confiabilidad es la capacidad del sistema para funcionar de manera consistente de acuerdo con sus especificaciones básicas. Tiene como objetivo aumentar la robustez de IoT para soportar problemas de seguridad, el uso a largo plazo, la funcionalidad de la aplicación al obtener información incierta, entre otros. Tiene como principal característica el garantizar la confiabilidad de la implementación en todas las capas de IoT. La red de IoT y la aplicación que se ejecuta en la red, deben lograr garantizar la transición la información de manera confiable.

- **Movilidad** Uno de los principales retos del IoT es el soporte a la movilidad IP entre las conexiones de los usuarios y los servicios deseados, en particular cuando se están moviendo en diferentes lugares y cuando ejecutan métodos de acceso. También es importante la detección de movimiento de los dispositivos IoT para la asignación de nuevas regiones o redes (Yaqoob et al., 2017).

- **Disponibilidad** La mayoría de las aplicaciones del IoT requieren de un alto nivel de disponibilidad de hardware y software para proporcionar servicios de manera eficaz. La mayoría de las aplicaciones del IoT

Implementación Ágil de Servicios para Aplicaciones de Internet de las Cosas

Erick Infante-Covarrubias * Regino Infante-Covarrubias * Javier Ortiz-Hernández *

* *Tecnológico Nacional de México/CENIDET Cuernavaca, Morelos, México (e-mail: {erick.infante19cn, rinfante19ce, ortiz}@cenidet.edu.mx).*

Resumen: El desarrollo de aplicaciones para el internet de las cosas tiene muchos desafíos que afrontar. El alto número de dispositivos del entorno físico que se conectan a la red producen cada vez una mayor cantidad de datos. Para la utilización de los datos es necesario el desarrollo ágil de servicios, el cual es una tarea difícil debido a la complejidad de los sistemas existentes. En este trabajo se muestra el uso de la plataforma de código abierto FIWARE para la implementación ágil de servicios que permitan explotar de manera adecuada los datos de ubicación en interiores para futuras aplicaciones del internet de las cosas.

Palabras Clave: Internet de las Cosas, IoT, FIWARE, NGSI

1. INTRODUCCIÓN

El internet de las cosas o IoT lo define (ITU, 2015) como "Una infraestructura global para la sociedad de la información que permite servicios avanzados interconectando elementos (físicos y virtuales) basados en tecnologías de la información y la comunicación existentes y en evolución".

Debido a la alta complejidad de estos entornos y a la falta de estandarización en las tecnologías se utilizan una amplia variedad de arquitecturas para su desarrollo dependiendo del tipo de entorno de la aplicación.

Existen dos arquitecturas de uso general para desarrollos en aplicaciones de internet de las cosas (Al-Qaseemi, Almulhim, Almulhim, & Chaudhry, 2016):

La arquitectura de tres capas (Figura 1): la cual consiste en las capas de percepción, red y aplicación. La capa de percepción contiene los dispositivos que recopilan datos y son enviados a la capa de red. La capa de red es la responsable de interconectar la capa de percepción con la capa aplicación. La capa de aplicación recibe los datos y los procesa para proveer los servicios solicitados.

Por otro lado, la arquitectura de cinco capas (Figura 2): es una adecuación de la arquitectura de tres capas a la que se le adicionan las capas de Gateway y Middleware. La capa Gateway es la responsable del manejo de dispositivos IoT y de efectuar el intercambio de mensajes entre los dispositivos y los diversos subsistemas. Por su parte la capa Middleware proporciona un vínculo flexible para la comunicación entre el conjunto de los dispositivos y las aplicaciones.

La plataforma FIWARE es una plataforma abierta y estandarizada basada en código abierto para fomentar el desarrollo de servicios y aplicaciones inteligentes en diferentes dominios del internet de las cosas (Martinez, Ramirez, Estrada, & Torres, 2017). Proporciona un conjunto de herramientas para integrar diferentes funcionalidades. Se define como un ecosistema de innovación para la creación de nuevas aplicaciones y servicios de Internet. Es especialmente útil para el desarrollo de aplicaciones Smart Cities, ya que asegura la interoperabilidad y la creación de modelos de datos estándar.

Uno de los componentes clave de FIWARE es el Orion Context Broker, ya que permite gestionar todo el ciclo de vida de la información de contexto, incluidas actualizaciones, consultas, registros y suscripciones. Por ejemplo, es posible efectuar suscripciones para recibir una notificación cuando algún elemento del contexto, que es de nuestro interés observar, ha cambiado (FIWARE, 2020b).

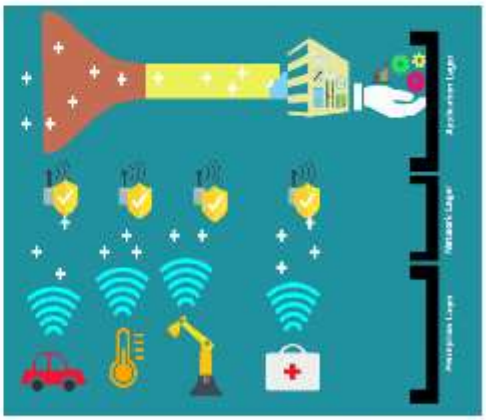

Fig. 1. Arquitectura de tres capas para IoT (Al-Qaseemi et al., 2016)

El objetivo de este trabajo es mostrar cómo una correcta implementación de servicios permite solventar los desafíos

Anexo B

Configuración de servidor IoT

FIWARE / DEVICEHIVE

ERICK INFANTE COVARRUBIAS

Contenido

Requerimientos 111
Instalación 113
Instalación de Ubuntu Server 22 113
Actualizar e instalar componentes adicionales 116
Instalar Docker-compose 118
Despliegue de FIWARE 118
Despliegue de DeviceHive 119

Requerimientos

Para el proceso de configuración del servidor IoT para ambas plataformas se utilizó un equipo con las siguientes características, 4 Gb de memoria RAM (Mínimo recomendado para DeviceHive), 10 Gb de almacenamiento mínimo, procesador de 4 núcleos (Mínimo recomendado para DeviceHive). Para la instalación se utilizó el sistema operativo Linux usando la distribución de Ubuntu Server 22.

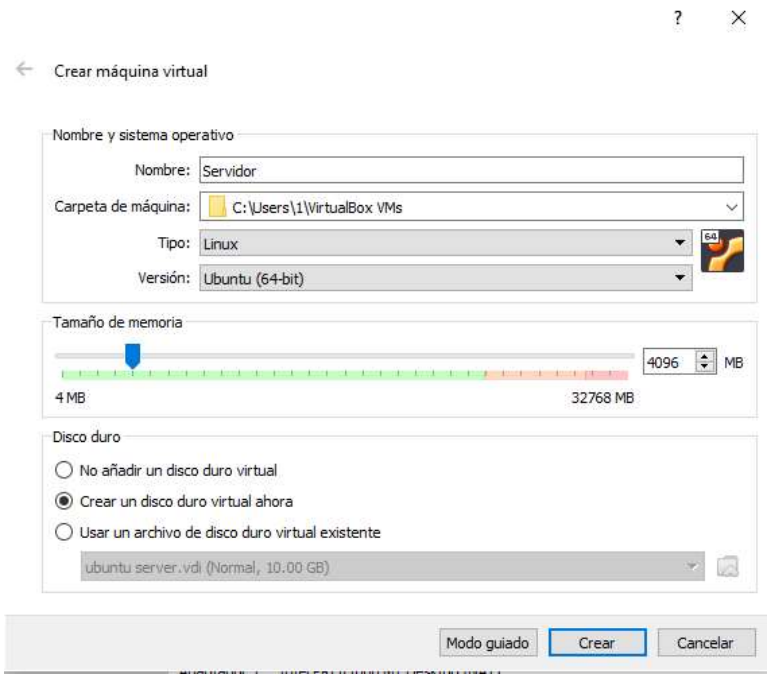

Figura 25 Configuración del servidor RAM

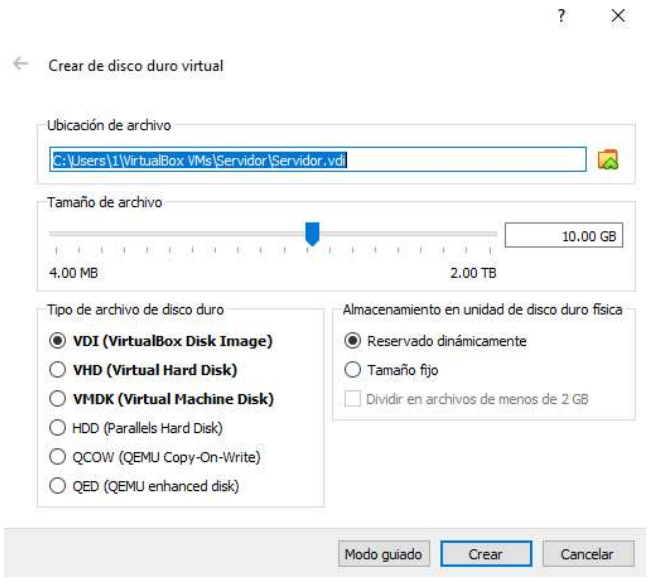

Figura 26 Configuración del servidor Disco Duro

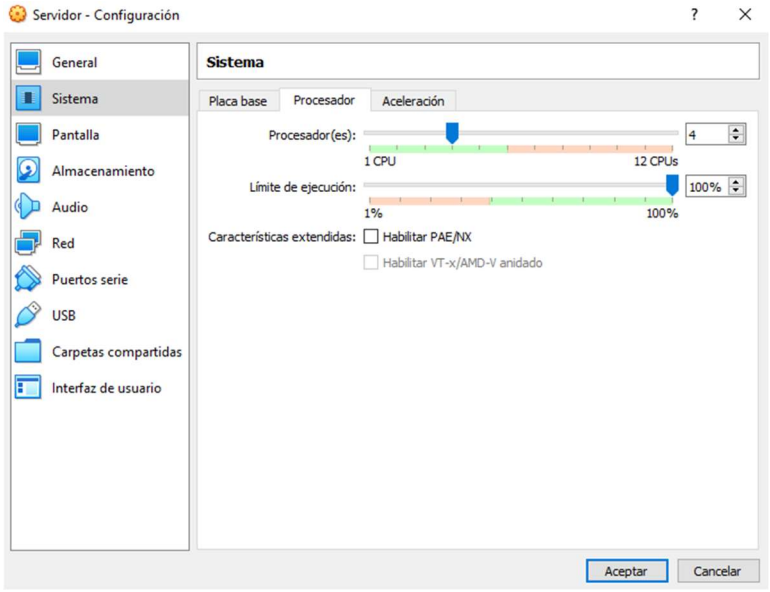

Figura 27 Configuración del servidor Procesador

Instalación

Instalación de Ubuntu Server 22

Ingresar la unidad con la imagen del sistema operativo (USB o DVD), al iniciar seleccionar la opción Instalar Ubuntu Server como se muestra en la figura 4. Después de que termine de cargar todos los recursos aparecerá un menú para seleccionar el idioma de la instalación, seleccionar el idioma español con las flechas y la tecla enter para confirmar la selección como se muestra en la figura 5.

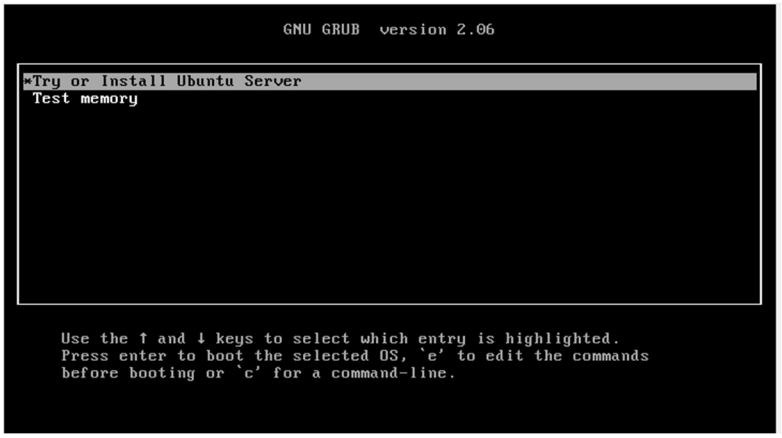

Figura 28 Instalar Ubuntu Server

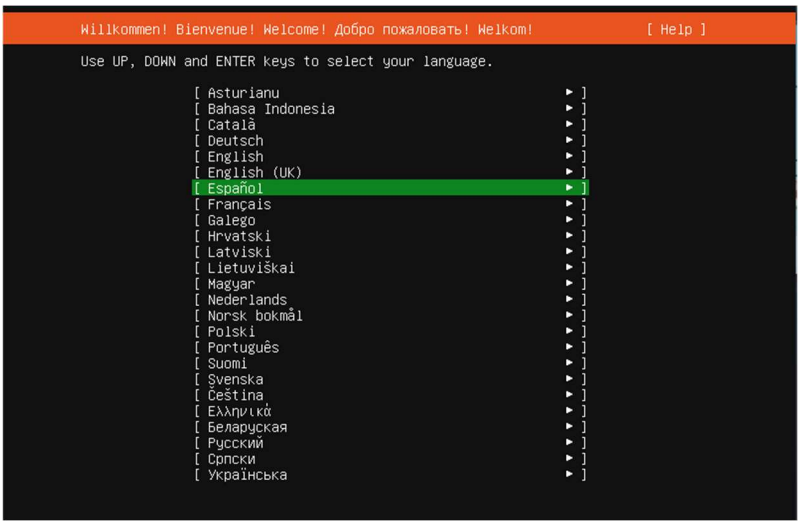

Figura 29 Seleccionar idioma

Una vez seleccionado el idioma se solicita la distribución del teclado que se está usando para la instalación, elegir el correspondiente y presionar enter. El siguiente paso es seleccionar la base de instalación (Normal o mínima), es recomendable seleccionar la opción completa

como se muestra en la figura 6. Una vez seleccionada si el servidor está conectado a la red detectará automáticamente la interfaz de red y una dirección IP de forma dinámica, la cual será necesaria para actualizaciones y conexión remota, en caso de necesitar colocar un proxy en la siguiente pantalla, dejar la dirección de repositorio de actualizaciones de Ubuntu que detecta automáticamente. En la pantalla siguiente se requiere seleccionar el uso completo del disco duro (en caso de no tener otro sistema operativo).

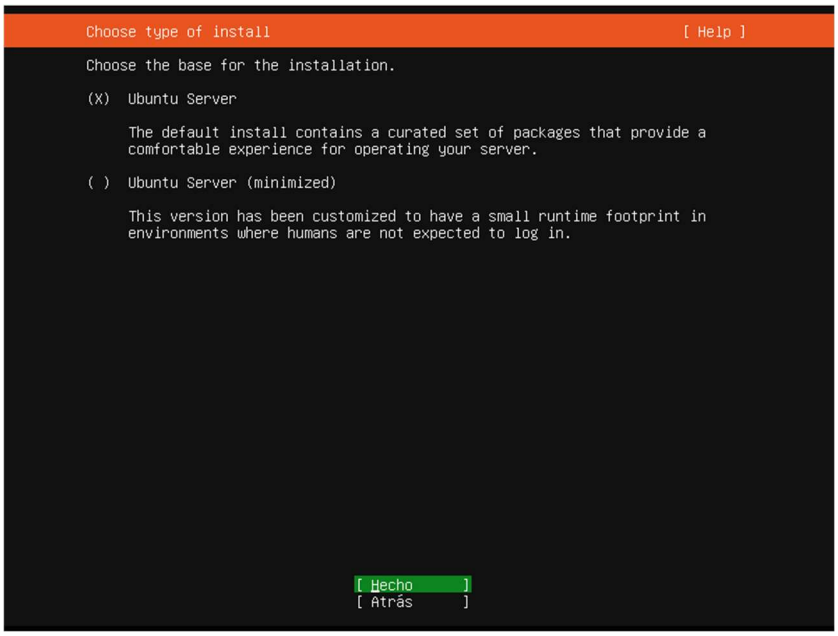

Figura 30 Base de instalación completa

Para finalizar con la instalación aparecerá una pantalla con un resumen de la configuración de instalación similar a la mostrada en la figura 7. Al seleccionar Hecho se pedirá una confirmación para evitar una configuración accidental por parte del usuario. Se pedirá ingresar datos de usuario para continuar con la instalación.

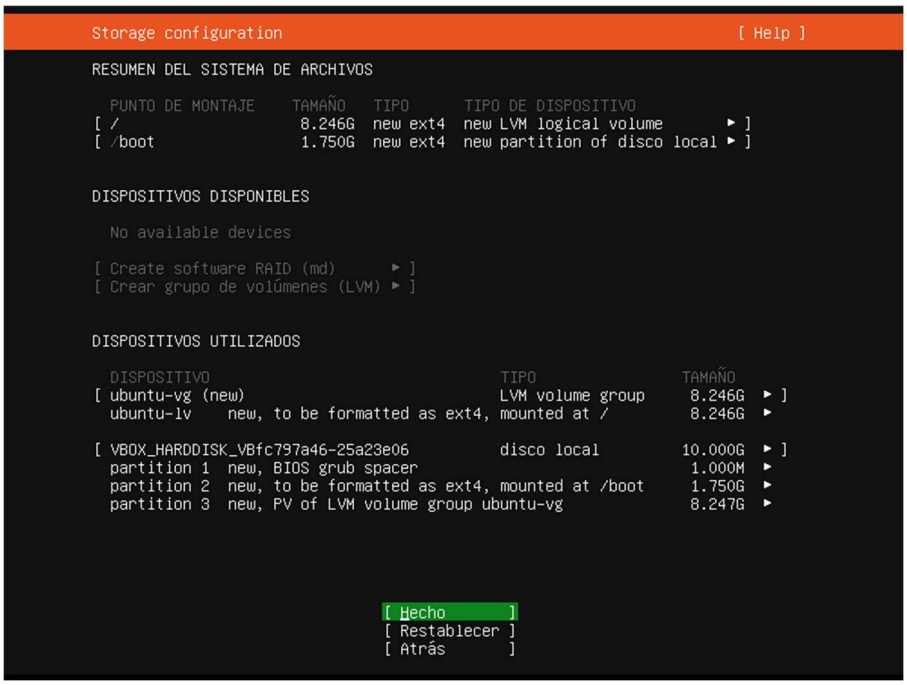

Figura 31 Resumen de configuración

En la figura 8 se muestra una pantalla donde da la opción de instalar un servidor de SSH para acceso remoto al servidor, para facilitar la configuración habilitar la opción y proceder con la instalación.

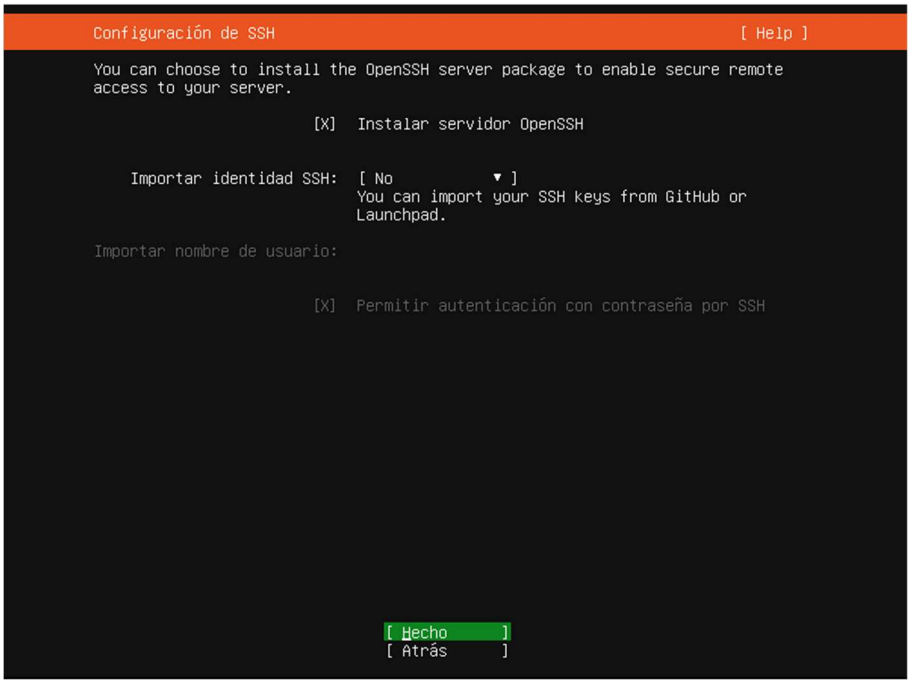

Figura 32 Instalación de servidor SSH

El siguiente paso de la instalación es seleccionar Docker como paquete opcional y proceder a la instalación como se muestra en la figura 9. Una vez completados los pasos aparecerá una ventana donde se mostrará el progreso de la instalación del sistema operativo y los componentes seleccionados, esperar a que termine la instalación, reiniciar el servidor, retirar la unidad con el sistema operativo e iniciar el servidor con el usuario y contraseña que se ingresaron durante la instalación.

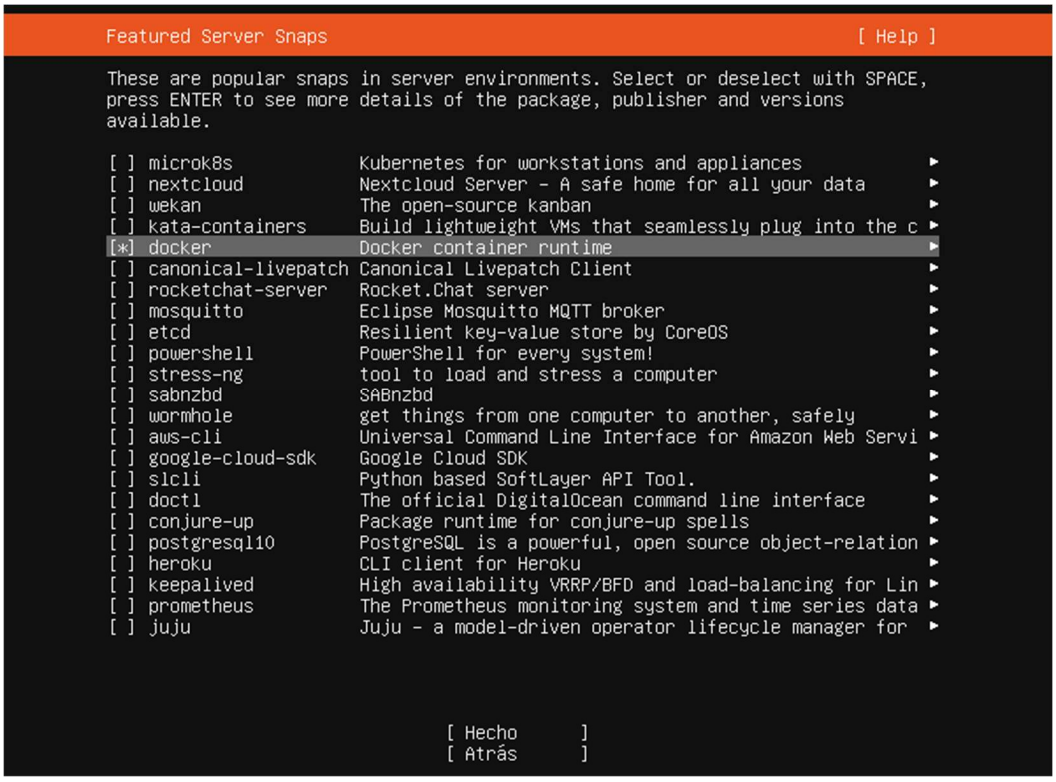

Figura 33 Instalación de Docker

Actualizar e instalar componentes adicionales

Lo primero es actualizar las dependencias del sistema operativo con los comandos “sudo apt update” y “sudo apt upgrade” como se muestra en la figura 10, el segundo comando pedirá una confirmación para instalar las actualizaciones. Y por último instalar las herramientas de red para consultar de manera fácil la IP con el comando “sudo apt install net-tools”.

```
root@iot:/home/r# apt update
Obj:1 http://mx.archive.ubuntu.com/ubuntu jammy InRelease
Des:2 http://mx.archive.ubuntu.com/ubuntu jammy-updates InRelease [114 kB]
Des:3 http://mx.archive.ubuntu.com/ubuntu jammy-backports InRelease [99,8 kB]
Des:4 http://mx.archive.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Des:5 http://mx.archive.ubuntu.com/ubuntu jammy/main Translation-es [332 kB]
Des:6 http://mx.archive.ubuntu.com/ubuntu jammy/restricted Translation-es [964 B]
Des:7 http://mx.archive.ubuntu.com/ubuntu jammy/universe Translation-es [1.356 kB]
Des:8 http://mx.archive.ubuntu.com/ubuntu jammy/multiverse Translation-es [68,2 kB]
Descargados 2.082 kB en 2s (984 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Se pueden actualizar 20 paquetes. Ejecute «apt list --upgradable» para verlos.
root@iot:/home/r# apt upgrade
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
Se actualizarán los siguientes paquetes:
  apt apt-utils cryptsetup cryptsetup-bin cryptsetup-initramfs dmidecode isc-dhcp-client
  isc-dhcp-common libapt-pkg6.0 libcryptsetup12 libldap-2.5-0 libldap-common libnftables1 nftables
  python3-distupgrade python3-software-properties software-properties-common
  ubuntu-advantage-tools ubuntu-release-upgrader-core zlib1g
20 actualizados, 0 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 4.400 kB de archivos.
Se liberarán 1.879 kB después de esta operación.
¿Desea continuar? [S/n] s
```

Figura 34 Actualización del sistema operativo

Una vez realizado la actualización, se recomienda una conexión SSH con el programa de Bitvise SSH Client (Figura 11), debido a que este cuenta a su vez con una conexión ftp para transferir archivos y se pueden copiar comandos directamente a la consola del servidor.

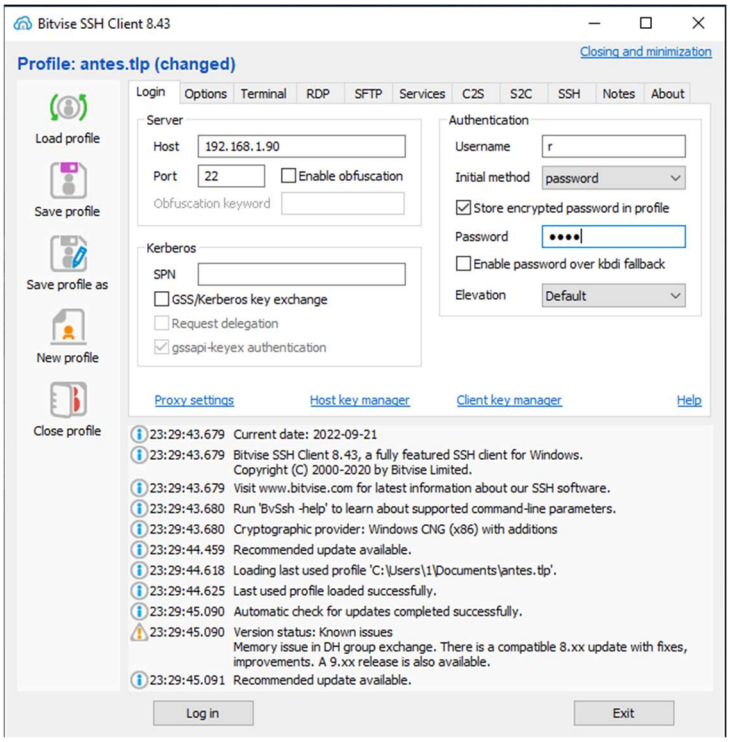

Figura 35 Bitvise SSH

Instalar Docker-compose

Para la instalación de la herramienta compose de Docker es necesario ejecutar los siguientes comandos en orden:

1. sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
2. sudo chmod +x /usr/local/bin/docker-compose
3. docker-compose –version

El primer comando descargara la versión 1.29 de la herramienta del repositorio de github y la colocara en el directorio /usr/local/bin/docker-compose. El segundo comando otorga permisos de ejecución al contenido del directorio /usr/local/bin/docker-compose. Y, por último, muestra la versión de la herramienta instalada. El resultado de ejecución de los comandos debe ser similar a la Figura 12.

```
root@iot:/home/r# sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-co
mpose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 12.1M  100 12.1M    0     0  4217k      0  0:00:02  0:00:02 --:--:-- 4857k
root@iot:/home/r# sudo chmod +x /usr/local/bin/docker-compose
root@iot:/home/r# docker-compose --version
docker-compose version 1.29.2, build 5becea4c
root@iot:/home/r#
```

Figura 36 Resultado de ejecución

Despliegue de FIWARE

El primer paso es descargar el archivo de configuración yml puede ser directamente del repositorio original (https://github.com/FIWARE/tutorials.Getting-Started) o descargar el archivo utilizado en este documento (https://github.com/ErickInCo/Tesis-Maestria/blob/09c503853f5a0065c0936dc6dd0d00e37a760a1e/Fiware%20y%20DeviceHive%20yml/docker-composeF.yml), copiar el archivo al servidor usando Bitvise, al estar en el mismo directorio del archivo ejecutar el siguiente comando para ejecutar Fiware

docker-compose -f docker-compose.yml up -d

Una vez terminado de descargar los componentes necesarios deberá mostrar un resultado similar a la Figura 13. Para asegurar que todo este corriendo de manera correcta al ejecutar el comando “Docker ps” deberá mostrar una salida similar a la Figura 14.

```
root@u:/home/e/fiware yml# docker-compose -f docker-compose.yml up -d
Creating network "fiwareyml_default" with the default driver
Pulling crate (crate:4.1.4)...
4.1.4: Pulling from library/crate
ab5ef0e58194: Pull complete
b82aa4130eb2: Pull complete
2c132278b93b: Pull complete
42807ee70d40: Pull complete
3a53fc6580d8: Pull complete
52fdb5ce574a: Pull complete
979fdf3de1dc: Pull complete
5edb73e27770: Pull complete
ec9ab38e5cad: Pull complete
a60b5292df9c: Pull complete
46096a9d919e: Pull complete
acd7d0076a91: Pull complete
Digest: sha256:3268623590124db93437f7d6a46ab8826d9d895abde796eebeaf2b33e2463482
Status: Downloaded newer image for crate:4.1.4
Creating fiwareyml_crate_1 ... done
Creating fiwareyml_mongo_1 ... done
Creating fiwareyml_redis_1 ... done
Creating fiwareyml_orion_1 ... done
Creating fiwareyml_grafana_1      ... done
Creating fiwareyml_quantumleap_1 ... done
root@u:/home/e/fiware yml#
```

Figura 37 Fiware en funcionamiento

```
root@u:/home/e/fiware yml# docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED         STATUS                   PORTS                                              NAMES
dafaed5fcc3e   smartsdk/quantumleap     "python app.py"          2 minutes ago   Up 2 minutes             0.0.0.0:8668->8668/tcp, :::8668->8668/tcp          fiwareyml_quantumleap_1
330fa8ecc50a   grafana/grafana          "/run.sh"                2 minutes ago   Up 2 minutes             0.0.0.0:3000->3000/tcp, :::3000->3000/tcp          fiwareyml_grafana_1
9b869752cf05   fiware/orion:2.0.0       "/usr/bin/contextBro…"   2 minutes ago   Up 2 minutes (healthy)   0.0.0.0:1026->1026/tcp, :::1026->1026/tcp          fiwareyml_orion_1
cd4a26781f75   redis                    "docker-entrypoint.s…"   3 minutes ago   Up 2 minutes             0.0.0.0:6379->6379/tcp, :::6379->6379/tcp          fiwareyml_redis_1
684d5f795ba5   mongo:3.2.19             "docker-entrypoint.s…"   3 minutes ago   Up 2 minutes             0.0.0.0:27017->27017/tcp, :::27017->27017/tcp      fiwareyml_mongo_1
root@u:/home/e/fiware yml#
```

Figura 38 Procesos Docker

Despliegue de DeviceHive

Para correr los componentes de DeviceHive en un servidor propio es necesario descargar los archivos del repositorio original en github (https://github.com/devicehive/devicehive-docker/tree/master/rdbms-image) o una copia en mi repositorio personal (https://github.com/ErickInCo/Tesis-Maestria/tree/main/Fiware%20y%20DeviceHive%20yml/DH), para después copiarlos en el servidor mediante FTP con Bitvise.

Para ejecutar DeviceHive es necesario estar en el directorio donde se copiaron los archivos, para después ejecutar el comando “docker-compose -f docker-compose.yml -f dh_plugin.yml up -d”, Docker comenzara a descargar e inicializar las instancias de los componentes necesarios para el funcionamiento de la plataforma como se muestra en la Figura 15. Para

revisar los puertos de cada componente ejecutar el comando “Docker ps” donde se mostrará el componente, así como el puerto designado como se muestra en la Figura 16.

```
root@u:/home/e/DeviceHiveFiles/rdbms-image# docker-compose -f docker-compose.yml -f dh_plugin.yml up -d
WARNING: The JWT_SECRET variable is not set. Defaulting to a blank string.
Creating network "rdbms-image_default" with the default driver
Creating rdbms-image_postgres_1  ... done
Creating rdbms-image_hazelcast_1 ... done
Creating rdbms-image_zookeeper_1 ... done
Creating rdbms-image_kafka_1     ... done
Creating rdbms-image_kafka-cleanup_1 ... done
Creating rdbms-image_wsproxy_1       ... done
Creating rdbms-image_dh_backend_1    ... done
Creating rdbms-image_dh_auth_1       ... done
Creating rdbms-image_dh_frontend_1   ... done
Creating rdbms-image_dh_plugin_1     ... done
Creating rdbms-image_wsproxyext_1    ... done
Creating rdbms-image_dh_proxy_1      ... done
root@u:/home/e/DeviceHiveFiles/rdbms-image#
```

Figura 39 DeviceHive en funcionamiento

```
root@u:/home/e/DeviceHiveFiles/rdbms-image# docker ps
CONTAINER ID   IMAGE                                  COMMAND                  CREATED         STATUS         PORTS                                                                                NAMES
5b07a0daed2e   devicehive/devicehive-proxy:3.5.0      "/bin/sh ./proxy-sta…"   3 minutes ago   Up 3 minutes   80/tcp, 8443/tcp, 0.0.0.0:80->8080/tcp, :::80->8080/tcp                              rdbms-image_dh_proxy_1
e6433cacc250   devicehive/devicehive-ws-proxy:1.1.0   "pm2-docker src/prox…"   3 minutes ago   Up 3 minutes   3000/tcp                                                                             rdbms-image_wsproxyext_1
ae6a91e1d86d   devicehive/devicehive-plugin:3.5.0     "/bin/sh ./devicehiv…"   3 minutes ago   Up 3 minutes   0.0.0.0:8110->8110/tcp, :::8110->8110/tcp                                            rdbms-image_dh_plugin_1
7eee1a6180df   devicehive/devicehive-frontend:3.5.0   "/bin/sh ./devicehiv…"   3 minutes ago   Up 3 minutes   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp                                            rdbms-image_dh_frontend_1
3b57b41495b7   devicehive/devicehive-auth:3.5.0       "/bin/sh ./devicehiv…"   3 minutes ago   Up 3 minutes   0.0.0.0:8090->8090/tcp, :::8090->8090/tcp                                            rdbms-image_dh_auth_1
f4eace6def29   devicehive/devicehive-backend:3.5.0    "/bin/sh ./devicehiv…"   3 minutes ago   Up 3 minutes                                                                                        rdbms-image_dh_backend_1
28d3119bfd83   wurstmeister/kafka:1.0.0               "bash -c 'while true…"   3 minutes ago   Up 3 minutes                                                                                        rdbms-image_kafka-cleanup_1
6cf97dcce84e   devicehive/devicehive-ws-proxy:1.1.0   "pm2-docker src/prox…"   3 minutes ago   Up 3 minutes   3000/tcp                                                                             rdbms-image_wsproxy_1
050c6459f31a   wurstmeister/kafka:1.0.0               "start-kafka.sh"         3 minutes ago   Up 3 minutes   0.0.0.0:9092->9092/tcp, :::9092->9092/tcp                                            rdbms-image_kafka_1
dbba78986e05   postgres:10                            "docker-entrypoint.s…"   3 minutes ago   Up 3 minutes   0.0.0.0:5432->5432/tcp, :::5432->5432/tcp                                            rdbms-image_postgres_1
2fdd81e0c5ef   devicehive/devicehive-hazelcast:3.5.0  "./server.sh"            3 minutes ago   Up 3 minutes   0.0.0.0:5701->5701/tcp, :::5701->5701/tcp                                            rdbms-image_hazelcast_1
d572b8b6d237   wurstmeister/zookeeper                 "/bin/sh -c '/usr/sb…"   3 minutes ago   Up 3 minutes   22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp, :::2181->2181/tcp                rdbms-image_zookeeper_1
root@u:/home/e/DeviceHiveFiles/rdbms-image#
```

Figura 40 Procesos de DeviceHive