Mecanismos Autónomos para la Autodescripción de Arquitecturas IoT Distribuidas

Henry Andres Jiménez Herrera

Trabajo de Grado para Optar por el Título de Magíster en Ingeniería de Sistemas e Informática

Director

Gabriel Rodrigo Pedraza Ferreira

PhD en Ciencias de la Computación

Universidad Industrial De Santander

Facultad De Ingenierías Fisicomecánicas

Escuela De Ingeniería De Sistemas E Informática

Programa de Maestría En Ingeniería De Sistemas E Informática

Bucaramanga

2022

Dedicatoria

A Dios todo poderoso por la fortaleza, habilidades y conocimiento para cumplir este objetivo en mi vida.

A mis padres por su apoyo incondicional y por el regalo de la vida.

Al profesor Gabriel Pedraza por sus enseñanzas, disposición y la orientación para el desarrollo de este proyecto.

A todos mis familiares, amigos y personas cercanas que brindaron parte de su tiempo para ayudarme, aconsejarme y darme alientos en el camino.

A los profesores que con sus enseñanzas me ayudaron a entender que siempre se puede ser mejor, y que siempre hay algo nuevo por descubrir.

A Henry Jiménez por no rendirse y perseverar en medio de la incertidumbre y las pruebas de la vida.

Tabla de Contenido

Pág.

Lista de Figuras

Pág.

Lista de Tablas

Pág.

Glosario

AC: autonomic computing (computación autónoma)

Cloud Computing: computación en la nube, provee servicios a gran escala mediante la infraestructura de Internet.

Edge Computing: computación de borde, paradigma de computación donde el procesamiento de la información se encuentra distribuido.

Fog Computing: paradigma de computación en la niebla también conocido como Fog Networking.

IoT: internet of things (internet de las cosas)

ITU: international telecommunication union conocida en español como UIT o Unión Internacional de Telecomunicaciones.

ODS: objetivos de desarrollo sostenible - ONU.

SDA: self-description adapter o adaptador de autodescripción, es uno de los mecanismos propuestos en el trabajo de investigación.

SDM: self-description mechanism o mecanismo de autodescripción, es un gestor autónomo propuesto en esta investigación para la autorrepresentación de componentes de la arquitectura IoT.

TIC: tecnologías de la información y la comunicación.

Resumen

Título: Mecanismos Autónomos para la Autodescripción de Arquitecturas IoT Distribuidas*

Autor: Henry Andres Jiménez Herrera**

Palabras Clave: Internet de las Cosas, Computación Autónoma, Arquitecturas Distribuidas, Autodescripción.

Descripción:

La gestión de sistemas IoT distribuidos es una tarea compleja, debido a la heterogeneidad y el dinamismo en los componentes del sistema y la escala de las aplicaciones desplegadas. Detectar de manera oportuna los cambios en los elementos software que componen una arquitectura IoT y compartir la información asociada a estos cambios correctamente es fundamental para realizar una administración eficiente. En este trabajo de investigación se ha propuesto una solución basada en computación autónoma que permite generar una representación de los elementos software que componen una arquitectura IoT. Para tal objetivo, primero: se definió un modelo conceptual que pudiera presentar una arquitectura IoT, segundo: se diseñó un conjunto de mecanismos autónomos que se encargan de monitorear, detectar, informar y analizar los cambios en el sistema para generar una representación del estado en tiempo cuasi-real basada en el modelo conceptual, esta representación es un repositorio de información para el sistema en sí mismo, que además provee una visualización orientada a grafos que permite a los administradores entender y gestionar mejor estos elementos, finalmente se valida el enfoque de la propuesta a través de la implementación de un prototipo que incluya los mecanismos descritos en la plataforma Smart Campus UIS y se realiza un conjunto de pruebas de coherencia, escalabilidad y rendimiento.

---

*Trabajo de investigación

**Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Gabriel Rodrigo Pedraza Ferreira, PhD. en Ciencias de la Computación.

Abstract

Title: Autonomous Mechanisms for Self-Description of Distributed IoT Architectures*

Author: Henry Andres Jiménez Herrera**

Key Words: Autonomous Computing, Internet of Things, Distributed Architecture, Self-Description.

DESCRIPTION:

The distributed IoT systems management is a complex task, due to the heterogeneity and dynamism of the system components and the scale of the deployment application, detecting on time the changes in the software elements that compose the IoT architecture and sharing the information associated with these changes correctly is fundamental for efficient management. This research proposes a solution based on autonomic computing that allows generating a representation of the software elements that compose an IoT architecture, for this purpose, first: a conceptual model that could present an IoT architecture was defined, second: a set of autonomous mechanisms are designed to monitor, detect, report, and analyze changes in the system to generate in near-real time a state representation based on the conceptual model. This representation is a repository of information for the system itself, which also provides a graph-oriented visualization that allows the administrator to better understand and manage these elements, finally, the proposed approach was evaluated through the implantation of a prototype that includes the mechanisms described in the Smart Campus UIS platform and a set of consistency, scalability and performance test are performed.

---

* Research Work

**Faculty of Physics-Mechanics Engineering. School of Systems Engineering and Computer Science. Advisor: Gabriel Rodrigo Pedraza Ferreira, PhD. In Computer Science.

Introducción

El acceso a las tecnologías de la información y la comunicación (TIC) es un esfuerzo global que está enmarcado dentro de los denominados objetivos de desarrollo sostenible (ODS) adoptados por la Asamblea General de las Naciones Unidas (Naciones Unidas, 2015), donde la meta para el 2030 es proporcionar acceso a internet al 100% de los hogares en países menos adelantados. El estado Colombiano -que es miembro de las Naciones Unidas- también ha enfocado en los últimos años sus esfuerzos por mejorar la cobertura y acceso a internet en el país, logrando aumentar la cobertura que para el 2015 era del 41.8% de los hogares colombianos, en el 2018 reportaba 49.9% como lo expone (Cervera, 2021) y el último dato disponible del DANE muestra un incremento para un acceso a internet del 51.9% en los hogares colombianos para el 2019 (Departamento Administrativo Nacional de Estadística – DANE, 2021), resaltando que la mayor parte de los hogares tienen acceso a internet mediante dispositivos como teléfonos inteligentes.

Debido al incremento de personas que tienen acceso a dispositivos de bajo costo con capacidades de cómputo y conexión, se ha incrementado la demanda de productos y servicios asociados a dispositivos conectados a internet, como lo reporta la plataforma Statista (Vailshery, 2022) el número de dispositivos conectados a internet será aproximadamente de 30 mil millones en el año 2030, lo cual estimula el desarrollo de nuevas plataformas que nos permiten reducir la brecha entre los entornos físicos y virtuales, es aquí donde podemos situar al Internet de las Cosas (IoT) por sus siglas en inglés, que por medio de la integración de entornos virtuales y reales permite generar soluciones en áreas como salud, agricultura de precisión, domótica, ciudades y campus inteligentes impactando sobre las empresas, hogares, universidades y ciudades en general.

La correcta adopción del IoT requiere una mezcla de conocimientos en las diferentes ramas de la ingeniería, entre las cuales se encuentra la ingeniería de software; en general los sistemas IoT agrupan un conjunto de dispositivos con capacidades de cómputo, almacenamiento, interacción, detección, acción y principalmente comunicación. Gran parte de estas capacidades son posibles gracias a los componentes software que soportan la infraestructura y actúan en los diferentes niveles de la arquitectura, pero diseñar, implementar, desplegar y administrar estas arquitecturas es una tarea compleja y tal como lo presenta (Čolaković & Hadžialić, 2018) los sistemas IoT poseen una infraestructura heterogénea con elementos software que tienen múltiples configuraciones y estados, esto dificulta su administración y la sincronización entre elementos, principalmente en arquitecturas distribuidas.

Todo esto representa una serie de retos que surgen a la hora de construir arquitecturas IoT altamente escalables en entornos distribuidos, la administración por ejemplo se convierte en una tarea compleja por la heterogeneidad de los componentes (hardware y software) presentes en la arquitectura, el carácter altamente distribuido de la infraestructura o las aplicaciones y el dinamismo presente en los dispositivos, servicios y demás componentes que entran, salen, fallan y en general cambian de estado continuamente. Entonces, la información de base que debería conocer un administrador es el estado del sistema, lo cual nos lleva a enfrentar otros retos asociados, por ejemplo, el mecanismo que debe usarse para monitorear la infraestructura, la representación que se debe asociar para mostrar el estado de los elementos que la componen, la sincronización de esta representación de forma que lo que se observa coincida con lo que está pasando en tiempo (cuasi) real en la arquitectura y adicionalmente, la comparativa del estado actual versus un estado deseado que pueda ser configurado en una representación dada y que permita identificar comportamientos o configuraciones no deseadas.

La visión que tienen los investigadores del área es crear sistemas más autónomos, que tengan capacidades de autogestión (totales o parciales), como lo describe (Alberti & Singh, 2013) el enfoque de la computación autónoma puede brindar soluciones a los problemas de administración en arquitecturas IoT, esta visión ya se encuentra presente en algunos de los productos que están disponibles en el mercado, por ejemplo, los asistentes digitales para el hogar como Alexa, tal y como lo estudian (Beneteau, y otros, 2019) tienen la capacidad de seguir instrucciones mediante reconocimiento de voz y realizar acciones sobre el entorno como encender o apagar dispositivos inteligentes conectados por protocolos como HTTP, MQTT, AMQP entre otros; pero el potencial de Alexa va más allá, puede detectar cuando un dispositivo no se encuentra en su estado normal, por ejemplo, cuando el usuario dejó la luz del estudio encendida, el sistema puede detectar que el dispositivo no se encuentra en su estado habitual para un horario determinado y puede decidir autónomamente acciones correctivas sobre el mismo.

El problema con dispositivos comerciales como Alexa, es que para los investigadores fuera de Amazon son cajas negras, muchas veces no es posible experimentar o investigar con estos dispositivos, o la mayoría de ellos están orientados a entornos con pocos elementos, como el hogar, que no están altamente distribuidos y que no requieren de un administrador para su correcto funcionamiento. En entornos más grandes como universidad o ciudades inteligentes, encontramos sistemas distribuidos conformados por diferentes subsistemas que se encargan de funcionar en contextos específicos, y que comparten la misma infraestructura, un ejemplo sería el subsistema de iluminación inteligente y otro el subsistema de monitoreo ambiental, estos subsistemas son básicamente un conjunto de elementos relacionados y asociados con una aplicación de las funcionalidades que proveen. De esta forma, los subsistemas conocen total o parcialmente la infraestructura donde funcionan y un administrador debería conocer: 1) los elementos que hacen

parte de la infraestructura y su estado. 2) Los subsistemas (o aplicaciones) que usan la infraestructura. 3) El impacto que genera un cambio sobre la infraestructura.

El trabajo de investigación aquí presentado aporta una solución al problema de representar la composición y los estados en arquitecturas IoT distribuidas, para esto se ha propuesto un modelo conceptual que permita describir los elementos de la arquitectura, también la inclusión de un conjunto de mecanismos con capacidades autónomas que permitieron la autodescripción de los elementos software y se obtuvo el estado del sistema en función de esta representación. El enfoque que se propuso logró un apoyo a la gestión de la arquitectura IoT con unos mecanismos lo suficientemente robustos, adaptables y total o parcialmente autónomos.

Los capítulos posteriores del documento presentan a detalle la investigación, en el capítulo uno se estudia el problema de investigación, en el capítulo dos se presenta los objetivos, el capítulo tres contiene el estado del arte, el capítulo cuatro presenta el marco teórico, luego el capítulo cinco presenta la metodología de investigación, en el capítulo seis se expone la representación o modelo conceptual propuesto, el capítulo siete presenta el diseño de mecanismos autónomos para representación del sistema IoT distribuido, el capítulo ocho contiene prototipado, pruebas y finalmente en el capítulo nueve se presentan conclusiones y el trabajo futuro.

1. Problema de Investigación

El desarrollo y la administración de arquitecturas distribuidas son tareas complejas, los ingenieros que trabajan en este tipo de sistemas se enfrentan a problemas más graves que las fallas informáticas típicas, “lo peor es que no siempre es posible saber si algo deja de funcionar en servicios distribuidos” (Gabrielson, 2019), por lo tanto, observar el estado actual de los elementos que componen una arquitectura se convierte en una necesidad para administradores y desarrolladores. El problema se vuelve más complejo para arquitecturas IoT distribuidas, debido a factores como la heterogeneidad, el dinamismo en los componentes del sistema y la escala de las aplicaciones desplegadas.

En la actualidad existen diversas plataformas que se basan en arquitecturas altamente distribuidas, un ejemplo de ello es la plataforma global conocida como Netflix (Wang, 2020) que tiene servidores en distintos países y que procesan grandes cantidades de solicitudes, esta empresa tiene una vasta experiencia y recursos para el monitoreo de su infraestructura, la cual se basa principalmente en la observación del estado en los elementos que componen la arquitectura. Pero estas plataformas poseen componentes específicos desarrollados o adaptados en su mayoría por la misma organización, además trabajan en un contexto bien definido, esto no es posible en la mayoría de los arquitecturas IoT distribuidas que son más complejas, fragmentadas y en constante evolución (Cheruvu, Kumar, Smith, & Wheeler, 2020), generalmente implican la interacción entre diferentes componentes (hardware y software) que son desarrollados por distintos equipos u organizaciones con visiones diferentes y que normalmente no están alineadas con el despliegue y la administración de la arquitectura IoT.

Las herramientas tradicionales para monitorear sistemas no son adecuadas en las arquitecturas IoT que tienden a ser altamente distribuidas, heterogéneas y se despliegan en escenarios muy dinámicos, esto muestra la necesidad de desarrollar herramientas específicas que se puedan adaptar y permitan abstraer la complejidad en estos sistemas. Adicionalmente es necesario considerar la escala de estas arquitecturas (Van Kranenburg & Bassi, 2012), debido a la naturaleza de sus componentes, específicamente pueden existir miles o incluso millones de dispositivos inteligentes activos que envían y reciben información de forma continua.

Para un administrador de arquitecturas IoT distribuidas sería inviable configurar manualmente cada uno de los dispositivos que son desplegados de manera que pueda ser monitoreado, incluso centrándose en configurar y observar los componentes principales de la arquitectura como nodos intermedios, no tendría claro la representación que debe usar con estos elementos, que pueda ser compatible con otros y al mismo tiempo pueda abstraer información fundamental de dispositivos tan heterogéneos, esto sumado a elegir entre los diferentes protocolos y tecnologías que se pueden llegar a encontrar implementados en la arquitectura y que no siempre son compatibles u óptimos para todos los problemas, este es uno de los principales desafíos en el desarrollo de arquitecturas IoT distribuidas (Čolaković & Hadžialić, 2018).

En consecuencia, desplegar arquitecturas IoT distribuidas sin un correcto mecanismo de observación, conlleva a distintos problemas de administración, como las demoras en la detección y reacción ante una falla del sistema, perdidas de tiempo del personal altamente capacitado que deberá emplearse en buscar y reparar estas fallas, así como generar mecanismo de monitoreo e intentar implementarlos en los componentes del sistema, estas acciones pueden llevar a conflictos en la configuración si no se implementa adecuadamente, y no será fácil realizar la evaluación de los cambios que se implementen en la arquitectura y su documentación.

Más allá de las dificultadas para los administradores de una arquitectura IoT, el problema es inherente a los componentes o dispositivos que la conforman, por su naturaleza heterogénea y dinámica, ya que en escenarios específicos un dispositivo puede dejar de funcionar correctamente al no poder comunicarse y operar con otros dispositivos, la interoperabilidad es una característica que toma relevancia en el mercado de dispositivos y plataformas IoT (Cheruvu, Kumar, Smith, & Wheeler, 2020). Por esta razón es necesario tener una representación que sirva no solo a los administradores del sistema sino también a los componentes que ingresan o se retiran de la arquitectura IoT permitiendo resolver en parte las brechas de interoperabilidad, entonces, es necesario estudiar mecanismos para la descripción de arquitecturas IoT distribuidas que brinden herramientas a los administradores y desarrolladores de manera que puedan enfrentar de una forma más eficiente los problemas anteriormente descritos y también a los componentes del sistema mismo para mejorar su interoperabilidad y compatibilidad.

1.1 Preguntas De Investigación

¿Qué debería considerar un modelo conceptual para representar los componentes de una arquitectura IoT altamente distribuida?

¿Cómo debería ser un mecanismo autónomo para la autodescripción de arquitecturas IoT distribuidas que use como base de conocimiento dicho modelo conceptual?

2. Objetivos

2.1 Objetivo General

Definir mecanismos de computación autónoma para la autodescripción de elementos software y sus relaciones en arquitecturas IoT distribuidas.

2.2 Objetivos Específicos

- Establecer un modelo conceptual para la descripción de elementos software y sus relaciones en arquitecturas IoT distribuidas.

- Definir los mecanismos autónomos que permitan la autodescripción de arquitecturas IoT distribuidas utilizando el modelo conceptual establecido.

- Validar los mecanismos autónomos definidos mediante la implementación de un prototipo o conjunto de escenarios de prueba.

3. Estado del Arte

Para abordar el estado del arte referente a la autodescripción de arquitecturas IoT distribuidas, se realizó la revisión de la literatura y se presenta desde lo más general a lo más específico con tres enfoques: el primero es el estudio de las propuestas de referencia de arquitecturas IoT que son usadas en la actualidad, el segundo enfoque es la búsqueda de trabajos relacionados con sistemas IoT que implementan algún componente de autonomía y el tercer enfoque se centra en los estudios relacionados con modelos e implementaciones que permitan la descripción de arquitecturas IoT finalmente se presenta una comparativa de los modelos conceptuales de referencia encontrados.

3.1 Arquitecturas IoT De Referencia

Uno de los principales retos a la hora de diseñar una arquitectura IoT es el establecimiento de un estándar que posea características como distribución, interoperabilidad y escalabilidad (Abdmeziem, Tandjaoui, & Romdhani, 2016), los tres modelos de arquitecturas más comúnmente usados son la arquitectura de tres capas, la arquitectura basada en servicios y las arquitecturas de software intermedio (Lombardi, Pascale, & Santaniello, 2021), existen modelos de hasta 7 capas como la arquitectura de referencia para IoT de (CISCO, 2014), la investigación presentada en (Mazon & Pan , 2021) muestra un análisis detallado de estos modelos, uno de los principales aportes que se puede resaltar de este trabajo es la identificación de los componentes en un ecosistema IoT, aquí en una arquitectura IoT se deben considerar elementos de la primera capa de percepción como sensores, actuadores y dispositivos inteligentes en general, también se deben considerar los elementos intermedios como tipos de redes de comunicación, protocolos, y los denominados Gateway IoT, finalmente los elementos que componen la capa de aplicación como

servidores, middleware, bases de datos, entre otros, también deben ser estudiados los diferentes paradigmas computacionales en torno a las arquitecturas IoT.

En el modelo de referencia de arquitectura IoT de tres capas, la primera capa se centra en los sensores, actuadores, transductores y diferentes dispositivos inteligentes, aquí el reto está asociado a la heterogeneidad de estos dispositivos (AI-Fuqaha, Guizani, Mohammadi, Aledhari, & Ayyash, 2015) que debe ser abordados en el diseño de arquitecturas IoT, algunas de las razones de dicha heterogeneidad es el origen de los dispositivos ya que son fabricados por distintas organizaciones, la multitud de lenguajes que existen para programarlos, los diferentes protocolos de comunicación existentes y la falta de estandarización (Al-Qaseemi, Almulhim, Almulhim, & Chaudhry, 2016).

La capa intermedia del modelo de tres capas también conocida como capa de transporte o capa de red se encarga de la transferencia de información desde los dispositivos hacía la capa de aplicación, en esta capa se pueden identificar componentes software, hardware, tecnologías y protocolos que permiten la conectividad entre los dispositivos IoT y la infraestructura Cloud. En (Čolaković & Hadžialić, 2018) se presenta una comparación entre los diferentes protocolos de comunicación y establece la necesidad de conocer el cómo y cuándo aplicar cada uno de ellos, adicionalmente se resalta la estandarización como "*la columna vertebral para el desarrollo de IoT en el futuro*".

En la última capa del modelo para arquitecturas IoT de tres capas, encontramos la capa de aplicación, la cual puede ser dividida por algunos autores para obtener el modelo de referencia de cuatro capas en arquitecturas IoT, descrita por (Liu, Fieldsend, & Min, 2017) que explica el procesamiento de datos y la comunicación entre nodos, la tercera capa que surge de esta división es la capa de procesamiento de datos que inicialmente se realizaría en nodos intermedios cómo los

gateways, laptops, y demás elementos con capacidades de cómputo no despreciables, estos elementos están muy relacionados con la denominada computación en la niebla (Fog Computing).

De los estudios relacionados respecto a esta capa se resalta el trabajo de (Sharma & Saini, 2019) quien propone sistemas distribuidos mediante la implementación de lo que denominan "Fog Clusters" que son nodos intermedios configurados como sistemas distribuidos para procesar la información, aprovechando adecuadamente las capacidades de los elementos que componen esta capa se pueden proponer diversas aplicaciones, tal es el caso de las presentadas por (Hajam & Sofi, 2021) donde se expone la importancia de usar adecuadamente los nodos de la Fog Computing, ya que representan disminución en costos de implementación y mejora en tiempos de respuesta, además se pueden encontrar muchos casos de uso relacionados con el área de la salud, transporte, redes de comunicación, redes de energía y demás contextos presentes en cuidades inteligentes.

La última capa del modelo de cuatro capas para arquitecturas IoT sería la capa cloud, middleware y de aplicaciones, aquí se localiza la infraestructura para la gestión de servicios, almacenamiento, procesamiento y las aplicaciones IoT, esta capa en las arquitecturas IoT debe brindar soluciones a las dificultades que expone (Dadkhah, Lagzian, Rahimnia, & Kimiafar, 2020) relacionadas con las grandes cantidades de datos que tienen que ser procesados y almacenados en los servidores o centros de datos, para esto se proponen arquitecturas en la nube distribuidas y acercar lo más posible el poder de procesamiento a la fuente de los datos, esto es el paradigma conocido como 'Edge Computing' (Bilal, Khalid, Edbad, & Khan, 2018) un dominio más amplio que generalmente requiere de la configuración de sistemas en la nube distribuidos y que permitan una comunicación rápida entre sus componentes, ya que a diferencia de las aplicaciones tradicionales de cloud computing en el Edge Computing se manejan volúmenes de datos más

grandes y se generan interacciones con más frecuencia, la noción de Edge Computing permite aprovechar elementos cercanos a la fuente de los datos con capacidades de procesamiento.

En la actualidad contamos con modelos de arquitecturas bien definidas para cada nivel de despliegue en arquitecturas IoT, y para la mayoría se puede encontrar que estarán basadas en estos modelos de referencia discutidos, a continuación, se estudian algunos trabajos que proporcionan comparativas entre diferentes soluciones IoT.

El trabajo de (Guth, Breitenbücher, Falkenthal, Leymann, & Reinfurt, 2016) presenta una comparativa entre arquitecturas open source y privadas, lo interesante de este trabajo es que tras realizar la comparativa también se propone una arquitectura IoT de referencia abstracta donde se pueden encontrar nuevamente conceptos muy relacionados a algunos de modelos ya presentados, lo cual lleva a pensar que existe un factor común o modelo de referencia que puede ser construido desde la abstracción para representar los elementos en una arquitectura IoT, sin embargo, algunas de las plataforma presentadas en esta investigación ya no están disponible, así que se hace necesario buscar una comparativa más reciente, esto nos lleva a grandes compañías como Microsoft, Amazon o Google que tienen departamentos enteros dedicados al desarrollo de productos relacionados con IoT, estas compañías están principalmente enfocadas en el desarrollo del Cloud Computing, y con base en su infraestructura proponen arquitecturas IoT de referencia, en el caso de Microsoft (Microsoft, 2022) esta arquitectura de referencia está orientada al uso de Azure IoT Central, una plataforma de aplicaciones que busca simplificar la creación de soluciones IoT.

Amazon propone un conjunto de productos y servicios (Amazon Web Services, 2022) para problemas de IoT, abarcan desde un sistema operativo para microcontroladores hasta servicios de análisis de grandes volúmenes de datos y optimizaciones de operaciones basada en la nube, luego

mediante la integración de estos servicios se construyen arquitecturas de referencia para casos de estudio específico como por ejemplo la arquitectura de referencia para el modelo de mantenimiento preventivo basado en machine Learning (Amazon Web Services, 2020). Otro referente es Google, pero realmente se presentan pocos productos orientados al IoT, el producto principal es IoT Core (Google, 2022) un servicio para conectar, administrar y transferir datos procedentes de dispositivos.

En este orden de ideas cada una de estas empresas tiene una propuesta de arquitectura diferente para trabajar con IoT, el trabajo presentado en (Pierleoni, Concetti, Belli, & Palma, 2019) permite tener una comparativa entre estas opciones, los objetivos principales de esta investigación se centraron en mapear los servicios ofrecidos por estas tres empresas, presenta métricas de comparación como gestión de dispositivos, protocolos de comunicación, reglas y analítica, almacenamiento de datos, integración, seguridad y costo de los servicios. El estudio concluye asegurando que es posible y viable conectarse de forma segura y privada a cualquiera de las tres plataformas, también determina que el protocolo de comunicación más usado es MQTT, las tres plataformas cumplen con las condiciones de calidad del servicio establecidas y no presentan una plataforma en específico como la más apta dejando esta decisión a los ingenieros que deban realizar despliegue de arquitecturas IoT sobre cualquiera de estos proveedores de infraestructura.

Con el análisis de los diferentes modelos y propuestas de arquitecturas IoT, así como las comparativas estudiadas, se puede determinar que es viable y (como se verá más adelante) muy factible determinar un conjunto de conceptos, propiedades y relaciones para representar la mayoría de los elementos presentes en las arquitecturas IoT actuales.

3.2 Arquitecturas IoT Autónomas

El paradigma de la computación autónoma fue presentado por IBM (Horn, 2001), esta propuesta busca dotar elementos computacionales con la capacidad de realizar determinadas tareas por sí mismos, esto permite reducir costes, tener sistemas más resilientes, escalables y administrables, lo cierto es ya tiene algún tiempo y en la actualidad encontramos nuevos enfoques como el paradigma de la computación autoconsciente (Lewis, Platzner, Rinner, Torresen, & Yao, 2016), sin embargo, la computación autónoma es un paradigma bastante más amplio y vigente que puede incluir gran parte de las propuestas actuales.

En las arquitecturas IoT se ve involucrado una gran cantidad de dispositivos que requieren implementación, configuración y administración de forma masiva, el mantenimiento de una arquitectura IoT de forma manual es inviable, por esta razón, se realizó una revisión de la literatura orientada a encontrar propuestas o soluciones a problemas en arquitecturas IoT que surjan desde el paradigma de la computación autónoma. El enfoque de esta revisión fue abarcar desde el contexto internacional y posteriormente reducirlo a propuestas nacionales y terminar con el estudio de propuestas locales.

En el panorama internacional se pueden encontrar diversos trabajos relacionados con la inclusión de autonomía en el IoT. El trabajo de (Thair, Ashraf, & Dabbagh, 2019) expone varios de los problemas que surgen en la administración de plataformas IoT, uno de ellos es la ubicación de los dispositivos, que normalmente se encuentran en un lugar remoto donde no es óptimo contar con personal para solucionar los problemas que se presenten de forma manual, adicionalmente debido a la escala y complejidad en las arquitecturas se espera que en algún momento la gestión de los dispositivos conectados supere las capacidades humanas, en dicho estudio se identifican algunas de las necesidades en arquitecturas IoT que pueden ser abordadas desde la computación

autónoma, se exponen tecnologías claves para su implementación y concluye presentando desafíos vigentes en este contexto de investigación, de estos desafíos se resaltan el diseño de sistemas de representación y toma de decisiones centralizados y descentralizados para arquitecturas IoT, también un desafío vigente es la implementación de los principios autonómicos en dispositivos IoT con recursos limitados, parte de estos desafíos serán abordados en este proyecto de investigación.

La gestión eficiente del consumo energético en los diferentes dispositivos IoT es otro de los desafíos existentes, (Sampaio, Koch, Becker, Boing, & Santa Cruz, 2019) presenta una posible solución a este problema proponiendo un mecanismo de gestión avanzada para optimizar el consumo de energía en entornos de fog computing, el enfoque propuesto permite visualizar la posibilidad de obtener un equilibrio entre la calidad del servicio y la demanda energética, en las pruebas lograron obtener ahorros de aproximadamente un 61% en el consumo de un dispositivo, la arquitectura propuesta tiene un fuerte fundamento lógico matemático y abarca muy bien las capas físicas e intermedias en una arquitectura IoT, pero no considera los elementos de la capa cloud como parte del modelo de autonomía.

El trabajo de (Abdellaoui, Megnafi, & Bendimerad, 2020) presenta un enfoque interesante para sistemas continuos o que no permitan la interrupción completa del servicio, el objetivo es lograr que sistemas IoT puedan autoconfigurarse en tiempo real sin la necesidad de dar de baja o reiniciar la totalidad de sus componentes, para esto usan una arquitectura basada en reglas que permite abordar mejor la incertidumbre en el conjunto de operaciones posibles del sistema. Proponen un metamodelo que abarca diferentes tipos de sistemas y que puede generar modelos específicos en función de un sistema deseado, es posible agregar o eliminar reglas en el sistema para lograr una adaptación al cambio, el trabajo propuesto como modelo de referencia es

interesante y podría complementarse con la implementación de la propuesta sobre una arquitectura IoT real. Otro enfoque para sistemas reconfigurables autónomos es el presentado por (Le-Dang & Le-Ngoc, 2019) donde se aprovecha la naturaleza distribuida y la seguridad de los sistemas basados en la tecnología del blockchain, esto permite desacoplar el procesamiento de novedades del sistema y facilitar los procesos de reconfiguración, en la investigación se evalúa la viabilidad de la propuesta y se logra determinar que la implementación de algunas tecnologías en arquitecturas IoT distribuidas tienen un impacto mínimo sobre el rendimiento de las mismas.

En el contexto nacional es posible encontrar algunos trabajos interesantes que usan IoT y computación autónoma, cómo es el trabajo de investigación presentado por (Alvarez, 2019) que está orientado a sistemas de control e implementación en la capa borde con microcontroladores, es uno de los trabajos encontrados a nivel nacional que profundizan en el área de IoT desde la computación autónoma y se resalta la implementación del ciclo MAPE-K dentro de un microcontrolador ESP32, logrando resultados interesantes en sistemas de control autónomos.

(Pico, Holgado, Sánchez, & Sampietro, 2018) presenta un enfoque denominado Internet de los Agentes (IoA, por sus siglas en inglés), este enfoque busca dotar de inteligencia y autonomía a los dispositivos que componen un ecosistema IoT, presenta 24 casos prácticos de aplicación de este enfoque que muestran cómo la inclusión de capacidades autónomas en sistemas IoT conlleva al desarrollo de sistemas inteligentes que requieren menos intervención humana para su gestión, nuevamente se resalta en las conclusiones de este trabajo la necesidad de estandarización y modelos de representación para los componentes de una arquitectura IoT, además de tecnologías abiertas que agilicen el desarrollo y la comunicación de los componentes del sistema.

Se realizó una búsqueda en las bases de datos de las universidades más grandes en Santander y aunque se encontraron muchos proyectos relacionados con modelos de arquitecturas

IoT y sobre todo casos de aplicación, sólo fue posible asociar a las áreas del IoT y la computación autónoma el trabajo de investigación que se presenta en (Jiménez, Cárcamo, & Pedraza, 2020), donde se da a conocer el modelo de referencia para la plataforma Smart Campus UIS basada en microservicios y se expone cierto grado de autonomía en términos de gestión, principalmente orientado a los componentes cloud mediante el despliegue distribuido de microservicios.

3.3 Descripción De Arquitecturas IoT

Aunque tiene algún tiempo sigue siendo vigente la propuesta de (IBM, 2006) donde se definen algunas características o condiciones que debe poseer un sistema autónomo, la primera de estas características es la capacidad de un sistema para conocerse a sí mismo o representarse de forma que pueda saber a qué recursos tiene acceso, qué elementos lo componen, cómo se conectan o interactúan estos elementos y las capacidades o limitaciones de estos, para representar los elementos de un sistema se generan algunos problemas, entre ellos el lenguaje o conjunto de conceptos que se usan para describir al sistema.

Un enfoque para abordar la heterogeneidad de los componentes en una arquitectura IoT es la generación de modelos de dominio que describan de forma abstracta estos componentes y sus relaciones, en la actualidad uno de los enfoques principales es el desarrollo de modelos conceptuales o modelos de dominio para el contexto de arquitecturas de sistemas, aquí podemos situar trabajos como el presentado por (Haller, Serbanati, Bauer, & Carrez, 2013) donde se propone un modelo de dominio orientado a los elementos que componen una arquitectura IoT, este modelo tiene en cuenta objetos animados como humanos, animales entre otros, hardware, software, y además nociones como realidad aumentada que son la mezcla entre entidades física y virtuales, de aquí se resalta la composición de dispositivos IoT tales como sensores y actuadores representados a manera de recursos asociados en una arquitectura IoT.

En el trabajo de (Negash, Westerlund, Rahmani, Lijieber, & Tenhunen, 2017) se presenta un modelo de dominio para sistemas IoT que responde a la complejidad del concepto de dispositivo y además cómo interactúan con entidades físicas, el metamodelo presentado es llevado a su implementación en un lenguaje de dominio específico llamado DoS-IL que permite describir entidades y relaciones conforme al metamodelo presentado. (Costa, Pires, & Delicato, 2019) presenta un lenguaje de modelado ampliado para descripción de aplicaciones IoT desplegadas en arquitecturas orientadas a servicios, el metamodelo presentado incluye la noción de servicio y aplicación, así como la forma de integrar estos conceptos en una arquitectura IoT orientada a servicios (SoA).

Trabajos como el de (Alulema, Criado, & Iribarne, 2019) presentan un metamodelo orientado al hardware presente en arquitecturas IoT, se tienen las nociones de servicios, sensores, actuadores y comunicación en general, con este trabajo surge la premisa de la no existencia en la literatura de un modelo asociado netamente al software que se despliega en una arquitectura IoT. En (González, Meana, García, Jiménez, & Petearson, 2020) se propone un lenguaje gráfico específico de dominio para modelar objetos inteligentes, el metamodelo presentado tiene en cuenta nociones como sensores, actuadores y las integra dentro de lo que denomina objeto inteligente, también se encuentra la noción de aplicación.

Dentro de los aportes más recientes se encuentra el presentado en (Alfonso, Garcés, Castro, & Cabot, 2021) donde se propone un lenguaje de dominio específico para representar características de una arquitectura IoT desplegada, lo interesante de este aporte es que abarca todas las capas de la arquitectura de referencia IoT y permite modelar adecuadamente los elementos hardware incluyendo el despliegue de nodos en contenedores con tecnologías como Kubernetes,

adicionalmente se presenta un mecanismo para expresar reglas de adaptación del sistema en función de dicha tecnología.

Finalmente, uno de los trabajos más recientes que podemos agregar al estado del arte es el aporte en (Erazo, Cedillo, Rossi, & Moyano, 2022) quien presenta un lenguaje de dominio específico con su respectivo metamodelo, el cual es bastante completo y permite representar los componentes de una arquitectura IoT con base en el estándar ISO/IEC 30141:218 (ISO, 2018), este trabajo fue publicado al mismo tiempo en el cual se terminó el desarrollo de este proyecto, por este motivo no pudo ser incluido dentro de las consideraciones del modelo conceptual desarrollado.

En función de la revisión de la literatura desarrollada fue posible determinar que el objetivo de esta investigación está alineado con las metas hacia las cuales están apuntando investigadores del área y puede generar un aporte para el desarrollo de los sistemas IoT autónomos del futuro.

3.4 Modelos Conceptuales De Referencia

Para el enfoque propuesto se buscó en la literatura diferentes modelos conceptuales relacionados con el IoT, luego se compararon los modelos más relevantes en función de la aparición o no de once conceptos que los autores consideran, hacen parte de las nociones que están presentes en los componentes software de una arquitectura IoT, estas nociones se definen en la Tabla 1 y permiten tener un punto de comparativa entre los modelos de domino encontrado.

Tabla 1

Conceptos Clave Arquitectura IoT

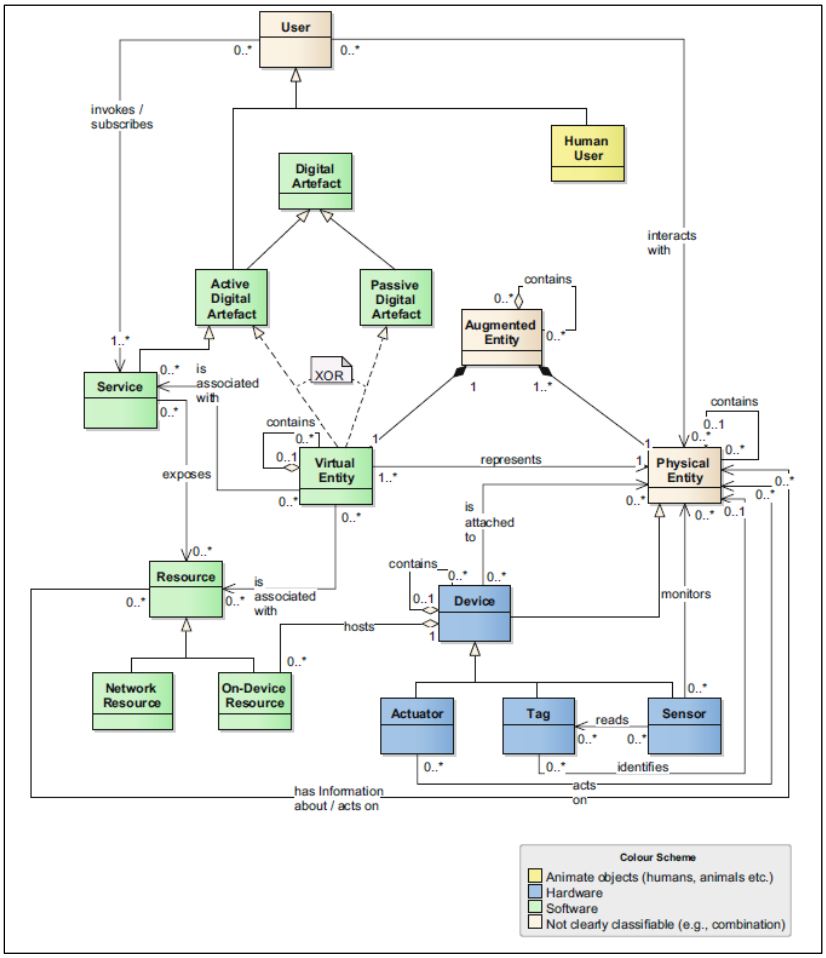
#	Concepto	Descripción
C1	Aplicación	Conjunto de dispositivos, sensores, actuadores, servicios y demás recursos interconectados que trabajan en función de un objetivo particular.
C2	Actuador	Elemento físico o virtual que puede generar una acción sobre el entorno.
C3	Almacenamiento	Elemento software que permite el almacenamiento de la información, por ejemplo, una base de datos.
C4	Comunicación	Elementos asociados a la transferencia de información entre elementos del sistema, ejemplo un bróker.
C5	Datos	Los datos recopilados por sensores, actuadores, servicios y demás elementos de una arquitectura IoT.
C6	Dispositivo	Entidad física o virtual que permite modelar la relación entre sensores, actuadores y demás recursos del sistema, puede tener sus propios atributos.
C7	Gateway	Elemento con capacidades de cómputo, comunicación, gestión, almacenamiento e integración con otros componentes de una arquitectura IoT.
C8	Propiedades	Características que definen el elemento que representan.
C9	Sensor	Elemento físico o virtual que permite recopilar información del entorno.
C10	Servicio	Representación lógica de un conjunto de actividades asociadas a un objetivo en particular de la lógica de negocios del sistema.
C11	Recurso	Elemento de la infraestructura IoT que provee algunas prestaciones.

Fuente. Elaboración propia

Dentro de los modelos conceptuales estudiados el punto de partida fue el modelo presentado por (Haller, Serbanati, Bauer, & Carrez, 2013) que muestra algunos de los conceptos presentes en la mayoría de los metamodelos encontrados en la literatura, en este modelo presentado en la Figura 1 podemos apreciar las nociones de servicio, recurso, dispositivos, sensores y actuadores, los autores muestran la versatilidad del modelo para representar ciertas configuraciones presentes en algunas aplicaciones IoT, sin embargo, algunas nociones como aplicaciones, almacenamiento, datos, no están presentes en este modelo.

Figura 1

Modelo conceptual de Haller, 2013



Nota: Tomado de *A Domain Model for the Internet of Things* (Haller, Serbanati, Bauer, & Carrez, 2013)

También se realizó la búsqueda y comparación de la literatura para encontrar modelos conceptuales que reuniera total o parcialmente los conceptos que se expresaron en la Tabla 1, con el objetivo de determinar qué tan común es encontrar dichas características en un modelo conceptual de una arquitectura IoT, y realizar la comparativa de los modelos más relevantes encontrados, los resultados de esta comparativa se resumen en la Tabla 2.

Tabla 2

Comparativa diferentes propuestas de modelos conceptuales

Propuesta	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
(Haller, Serbanati, Bauer, & Carrez, 2013)	NO	SI	NO	SI	NO	SI	NO	NO	SI	SI	SI
(Negash, Westerlund, Rahmani, Lijieber, & Tenhunen, 2017)	NO	SI	NO	SI	NO	SI	NO	SI	SI	SI	SI
(Costa, Pires, & Delicato, 2019)	SI	NO	NO	SI	NO	SI	NO	SI	NO	SI	SI
(Alulema, Criado, & Iribarne, 2019)	NO	SI	NO	SI	SI	NO	NO	NO	SI	SI	NO
(González, Meana, García, Jiménez, & Petearson, 2020)	SI	SI	NO	SI	NO	SI	NO	NO	SI	NO	NO
(Alfonso, Garcés, Castro, & Cabot, 2021)	SI	SI	NO	SI	NO	SI	SI	NO	SI	NO	NO

Nota: C1-Aplicación, C2-Actuador, C3-Almacenamiento, C4-Comunicación, C5-Datos, C6-Dispositivo, C7-Gateway, C8-Propiedades, C9-Sensor, C10-Servicio, C11-Recurso.

A través de esta revisión fue posible determinar que en algunos de los modelos actuales se omite la representación de elementos software importantes como bases de datos, la noción de Gateway, aunque aparece también como nodo de fog computing, el mapeo de los datos generados por los componentes dentro del modelo no es frecuente, y las nociones más comunes son sensores, actuadores y dispositivos.

4. Marco Teórico

En esta sección se resumen algunos de los fundamentos teóricos asociados al tema de investigación, inicialmente se presenta la visión, modelo de capas y paradigmas asociados al IoT, luego se exponen los conceptos principales de la computación autónoma, seguido por algunos conceptos principales de arquitecturas distribuidas, finalmente se presentan las nociones de modelo conceptual y grafo de conocimiento.

4.1 Internet De Las Cosas (IoT)

Una de las primeras declaraciones del Internet de las cosas (IoT) como un nuevo concepto es la de (Ashton, 2009) en 1999, posteriormente en el 2005 la Unión Internacional de Telecomunicaciones (UIT) presentaría un reporte (International Telecommunication Union, 2005) donde se expuso la importancia a futuro del potencial el desarrollo de los sistemas IoT, algunos de los retos que se debían abordar en este nuevo enfoque y que fueron objetivo de investigación en los años posteriores.

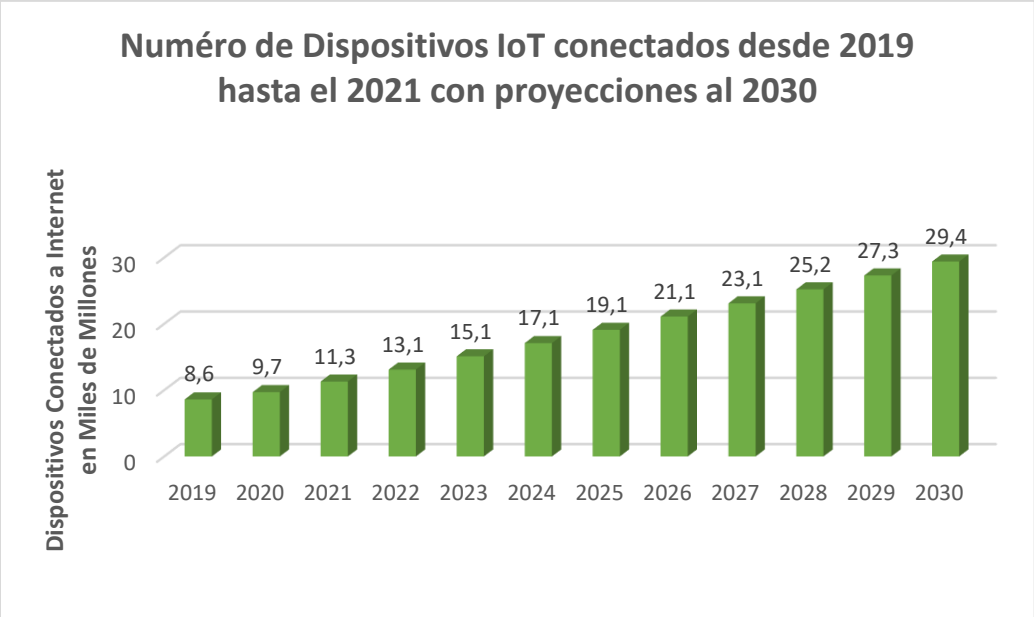
Pero ¿Qué es el IoT?, básicamente desde que apareció internet las redes de comunicación y la Web en sí misma ha estado evolucionando, las primeras versiones de la web solo permitían la comunicación entre personas, estas redes también conocidas también como redes Peer to Peer o P2P son el modelo de comunicación más antiguo que existe entre ordenadores, las redes han evolucionado a tal punto que hoy en día se tienen comunicaciones entre máquinas de forma autónoma y sin intervención humana, ahora bien, esta característica estaba reservada principalmente a equipos de cómputo o servidores hasta hace un par de décadas cuando se empezó a dotar a pequeños elementos de acceso a internet, gracias a los avances tecnológicos en materia de comunicación y microcontroladores, hoy en día es normal encontrar elementos cotidianos como

televisores, neveras, relojes y demás dispositivos conectados a internet, la capacidad de dotar a cualquier elemento de la vida cotidiana es lo que conocemos como el IoT o Internet de las Cosas y este paradigma está revolucionando la forma que tenemos de interactuar con nuestro entorno.

La gran acogida de las tecnologías del IoT ha generado grandes retos que deben ser abordados desde las ciencias de la computación, como se aprecia en la Figura 2 el número de dispositivos conectados a internet es creciente y la tendencia no parece que vaya a disminuir en la próxima década, de hecho aproximadamente en el año 2008 el número de dispositivos conectados a internet superó la población mundial, en el 2022 esta cifra sigue en aumento llegando a más de 13 mil millones de dispositivos.

Figura 2

Proyección del número de dispositivos conectados a internet

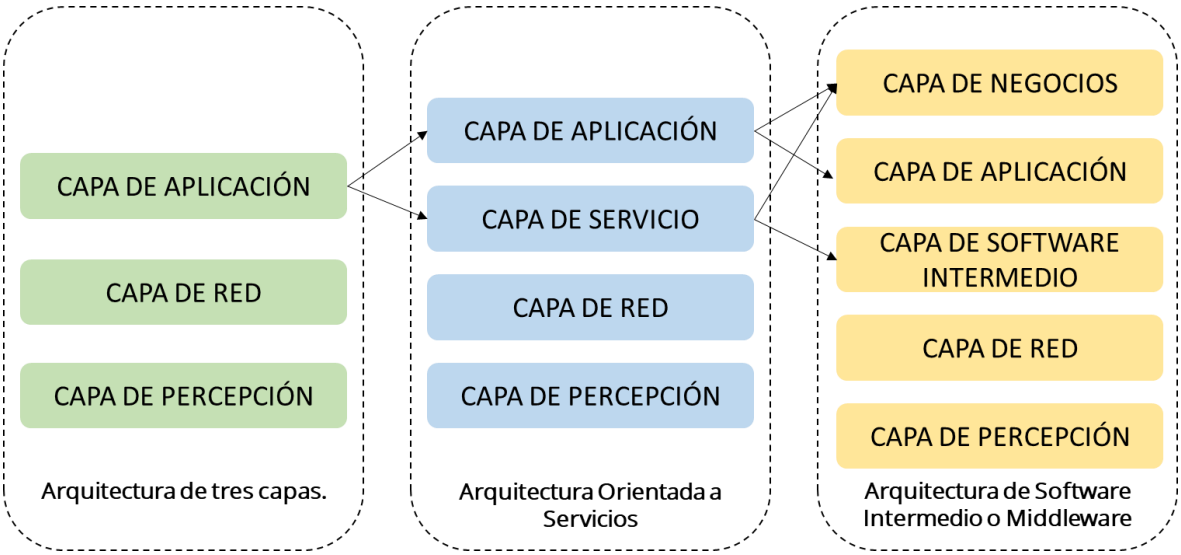


Nota: Datos tomados de (Vailshery, 2022)

Para brindar soluciones que puedan estar a la medida de la cantidad de dispositivos y elementos que conforman un ecosistema IoT se han propuesto varias arquitecturas de referencia, algunas de las más comunes pueden apreciarse en la Figura 3, donde se ve el clásico modelo de arquitectura de tres capas, luego la arquitectura de cuatro capas orientada a servicios y finalmente la arquitectura de cinco capas de software intermedio o Middleware.

Figura 3

Modelos de referencia de las Arquitecturas IoT más comunes



Nota: Adaptado de (Lombardi, Pascale, & Santaniello, 2021)

4.1.1 Arquitectura de Tres Capas

La arquitectura de tres capas fue introducida en la literatura por (AI-Fuqaha, Guizani, Mohammadi, Aledhari, & Ayyash, 2015) las capas descritas en esta arquitectura son:

Capa de percepción: Esta capa representa los objetos físicos de la arquitectura, aquellos dispositivos con distintos sensores y actuadores que se comunican directamente con el entorno, son la base del IoT y se llaman dispositivos inteligentes por su capacidad de conectarse con otros

dispositivos en red y sus propiedades de identificación, diagnóstico, reporte de fallas, etc. Un ejemplo de los elementos que pueden componer esta capa puede ser un televisor, un aire acondicionado o inclusive elementos sencillos como un sensor conectado a una placa de bajo coste que posee capacidades computacionales.

Capa de red: La capa de red se encarga del transporte de datos generados por los dispositivos en la capa de percepción, el catálogo de protocolos y tecnologías usados en IoT es bastante amplio y deben ser implementados conforme al tipo de aplicación y a las capacidades de cada dispositivo.

Tabla 3

Resumen de protocolos usados en el IoT

Capa de enlace de datos	Capa de adaptación	Capa de red	Capa de Transporte	Capa de Aplicación
Ethernet		**Dirección:**	TCP	CoAP
Wi-Fi	6LoWPAN	IPv4/IPv6	UDP	MQTT
Bluetooth	6TiSCH	**Enrutamiento:**		AMQP
RFID, NFC	6Lo	RPL		XMPP
ZigBee		CORPL		DSS
Z-Wave		CARP		

Capa de Aplicación: En esta capa están incluidos todos los servicios y el software necesario para el procesamiento y análisis de la información proveniente de las demás capas, en esta capa se puede situar los servidores en la nube, sistemas de bases de datos, y demás componentes con grandes capacidades de almacenamiento y procesamiento.

4.1.2 Arquitectura Orientada a Servicios

La arquitectura orientada a servicios o SoA permite coordinar servicios y hacer un uso más eficiente del software y hardware disponible en la arquitectura, básicamente es una extensión de la arquitectura de tres capas presentada anteriormente donde se incluye una capa adicional

conocida como la *capa de servicio*, se encarga de brindar apoyo a la capa de aplicación mediante funcionalidades como el descubrimiento de servicios, composición y gestión de servicios y las interfaces de acceso.

4.1.3 Arquitectura de Software Intermedio o Middleware

La arquitectura IoT basada en middleware o arquitectura de cinco capas (Ngu, Gutierrez, Metsis, Nepal, & Sheng, 2016) es un modelo de referencia que mezcla las capas de servicio y la capa de aplicación del modelo de referencia anterior para generar dos capas específicas, de esta forma se consigue una gestión más eficiente de las aplicaciones desplegadas en la arquitectura IoT. La capa middleware o de software intermedio tiene funciones de agregación y filtrado de datos, descubrimiento de información y control de acceso a los dispositivos por partes de las aplicaciones, en resumen, esta capa gestiona el acceso entres dispositivos y aplicaciones.

4.1.4 Paradigmas de computación presentes en el IoT

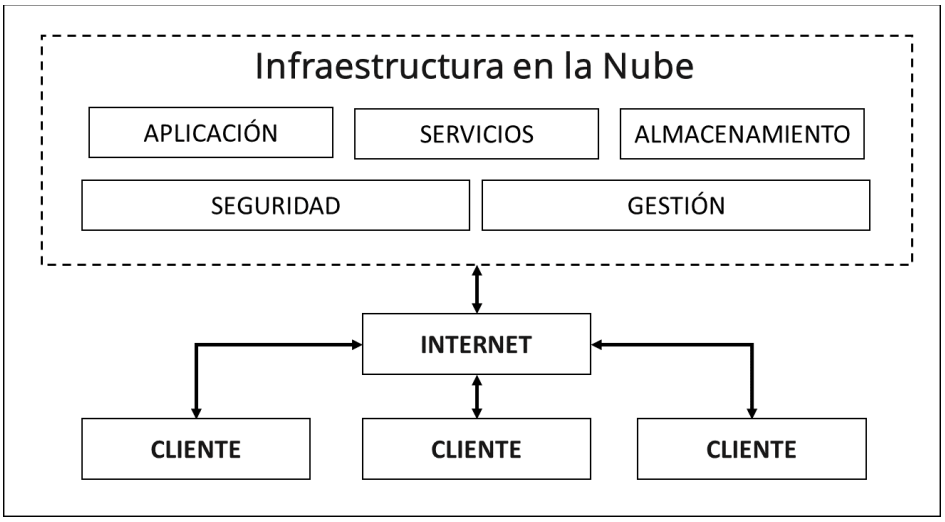
Para el desarrollo de las tecnologías de IoT se ha aprovechado paradigmas de computación existentes e incluso se han propuesto nuevos paradigmas, estos modelos establecen la forma en la cual es posible diseñar arquitecturas IoT que respondan a las necesidades actuales, y son un punto de referencia para los diseñadores de arquitecturas, a continuación, se definen brevemente estos paradigmas.

Computación en la nube (Cloud Computing): Se puede definir a la computación en la nube como el acceso a recursos computacionales de terceros como almacenamiento, procesamiento, comunicación, gestión de recursos y demás servicios a través de internet (Senyo, Addae, & Boateng, 2018), permitiendo a las empresas reducir costos y evitar tener que crear su propia infraestructura.

En el cloud computing existen cuatro modelos de implementación, nubes públicas, nubes privadas, nubes híbridas y nubes comunitarias, esta clasificación está en función de quién tiene acceso a los recursos, por ejemplo las nubes privadas son centros de computación que pertenecen a una sola organización, además existen tres modelos de prestación del servicio conocidos como software como servicio (SaaS), plataforma como servicio (PaaS) e infraestructura como servicio (IaaS) , en la Figura 4 podemos observar el esquema general del paradigma de computación en la nube donde los clientes acceden a los recursos que proporcionan proveedores cloud como Amazon, Microsoft o Google a través de internet.

Figura 4

Computación en la Nube - Cloud Computing



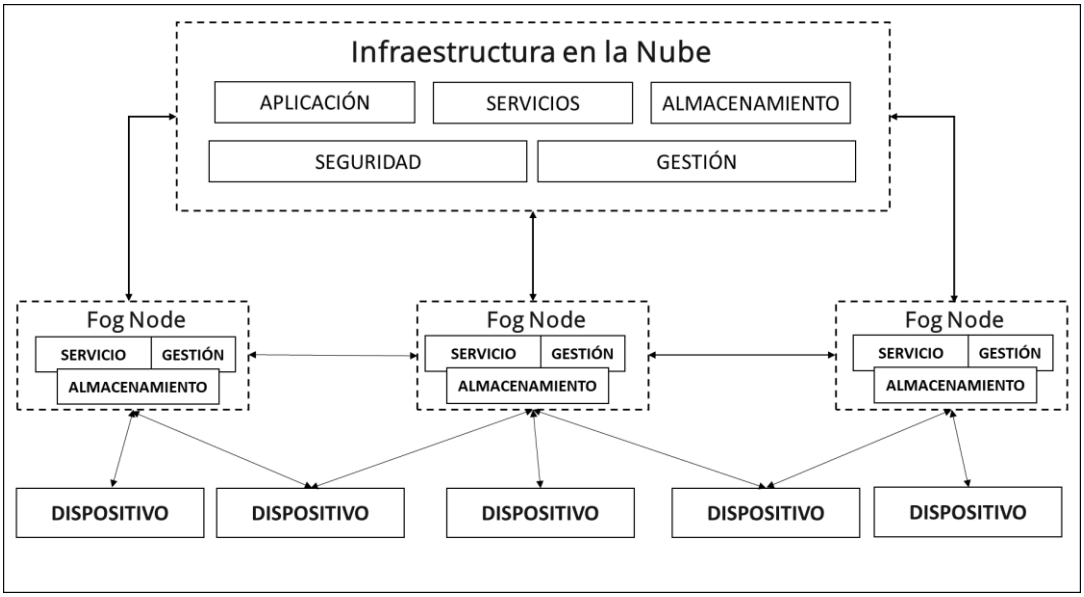
Computación en la niebla (Fog Computing): Existen contextos de aplicación donde no tenemos acceso a redes de comunicación tan estables, sobre todo en sistemas IoT donde una fábrica inteligente puede estar por ejemplo en una zona rural, en estos contextos tener todo el sistema basado en tecnología en la nube representa un problema y para esto se propone el paradigma de la computación en la niebla o fog computing, aquí se descentralizan algunos de los servicios que

proporciona la nube como almacenamiento o procesamiento de los datos y se distribuyen en una capa intermedia, en esta capa intermedia podemos encontrar micro centros de datos que contienen subsistemas de almacenamiento, gestión y procesamiento (Aazam, Zeadally, & Harras, 2018).

El modelo de computación en la niebla es una de las propuestas actuales que más aceptación tienen para la construcción de arquitecturas IoT, como se observa en la Figura 5 permite tener nodos de procesamiento más cerca de los dispositivos y manejar distintos protocolos de comunicación, estos nodos pueden ser incluso dispositivos de bajo costo como tarjetas Raspberry permitiendo abarcar mejor la heterogeneidad de los dispositivos IoT y generar arquitecturas distribuidas más estables y escalables.

Figura 5

Computación en la Niebla - Fog Computing



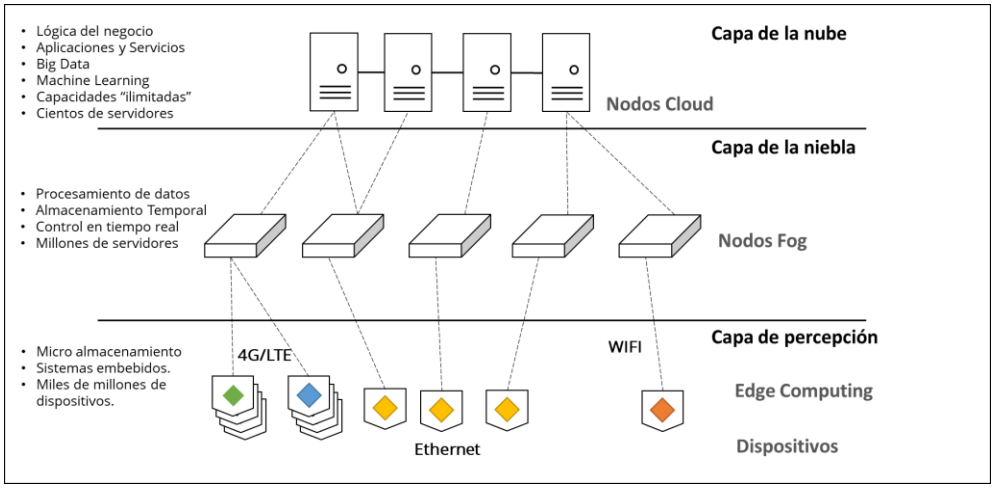
Los nodos Fog o nodos intermedios deben encargarse del enlace entre aplicaciones y dispositivos, el preprocesamiento de datos y el almacenamiento temporal de información, también

pueden ser usados en la gestión de los dispositivos IoT, pueden incluso interactuar y sincronizarse entre ellos para producir clúster como los propuestos por (Sharma & Saini, 2019).

Computación de borde (Edge Computing): Aunque la propuesta del fog computing permite generar arquitecturas IoT distribuidas y respuestas más rápidas, sigue existiendo el factor influyente de la comunicación entre dispositivos y los nodos fog, dependiendo de qué tan buena sea la conexión la respuesta se acerca más a lo que llamaríamos en tiempo real, ahora el escenario donde acercamos el poder de procesamiento lo más posible a la fuente de los datos o los dispositivos es lo que se conoce como Edge Computing o Computación de Borde, como el procesamiento ocurre más cerca de la fuente de los datos la latencia es reducida significativamente y tenemos mayor velocidad en las respuestas, los dispositivos Edge pueden ser procesadores de bajo costo, dispositivos como teléfonos inteligentes, inclusive equipos de cómputo que se encuentran desplegados en la misma área de los dispositivos IoT, la Figura 6 presenta el paradigma de la computación de borde, aquí se puede apreciar que las capacidades de cómputo, procesamiento y almacenamiento están presentes en todas las capas de la arquitectura.

Figura 6

Computación de Borde - Edge Computing



Nota: Adaptado de (Mazon & Pan , 2021).

La comprensión de los tres paradigmas anteriormente expuestos es fundamental en el diseño de cualquier arquitectura IoT, actualmente se pueden encontrar nuevos paradigmas como el Social Computing (Evans, 2020) que no serán abordados en este trabajo.

4.2 Computación Autónoma

El paradigma de la computación autónoma (llamada también AC por sus siglas en inglés) puede verse como una mezcla entre teoría de control y teoría de sistemas, busca dotar a los sistemas de cierto grado de autonomía, permitiendo que se autogestionen, adapten al cambio, corrijan fallos y en general que realicen de forma autónoma tareas que en general requieren de intervención humana. El concepto fue presentado por IBM (Horn, 2001) como una respuesta ante la preocupación por la creciente complejidad de los sistemas, el paradigma expone ventajas como la reducción de errores, costos de mantenimiento y en general administración de sistemas.

En 2004 se presentó una guía para el desarrollo de sistemas autónomos (Jacob, Lanyon-Hogg, Nadgir, & Yassin, 2004) donde se establecieron ocho condiciones que definen a un sistema autónomo:

I. El sistema debe conocerse a sí mismo, sus recursos, capacidades, limitaciones y cómo está conectado a otros sistemas.

II. El sistema es capaz de configurarse y reconfigurarse de forma autónoma según los cambios en su entorno.

III. El sistema puede optimizarse a sí mismo garantizando un rendimiento y uso de recursos eficiente.

IV. El sistema debe ser autoadaptable, dicho de otra forma, ser capaz de resolver un problema que surja mediante reconfiguraciones o aislamiento de funciones.

V. El sistema debe ser capaz de interactuar con los sistemas adyacentes, haciendo uso de los protocolos y mecanismos de comunicación de que disponga para adaptarse a los cambios de su entorno.

VI. El sistema debe poder establecer comunicaciones mediante estándares abiertos y no deben existir fronteras por componentes patentados y/o privativos.

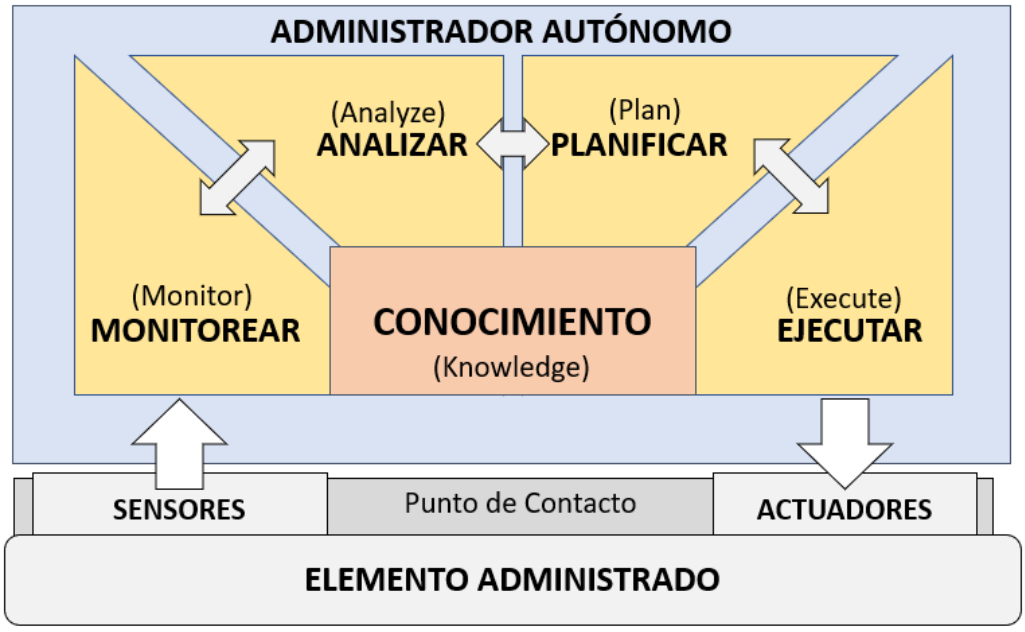
VII. El sistema debe anticiparse a las condiciones de su entorno y adaptarse en función de ello, por ejemplo, para suplir la demanda de un servicio en el tiempo.

En la actualidad es posible encontrar sistemas que ya cumplen con estas propiedades o parte de ellas, pero para entender mejor cómo se propone lograr la implementación de sistemas autónomos el punto de referencia sería el estudio del concepto de gestor o administrador autónomo,

un gestor autónomo es un mecanismo que realiza un conjunto de procesos que le permiten actuar de manera inteligente sobre un elemento administrado, la base de su funcionamiento es el bucle o ciclo conocido como MAPE-K de *Monitoring, Analyze, Plan, Execute – Knowledge*, la Figura 7 presenta el modelo general de un gestor autónomo, cada uno de los procesos que realiza tienen una secuencia definida.

Figura 7

Modelo general de un Administrador Autónomo



De acuerdo con (Jacob, Lanyon-Hogg, Nadgir, & Yassin, 2004) podemos definir cada uno de los elementos y procesos de un administrador autónomo de la siguiente manera:

- **Elemento Administrado**: Hace referencia al elemento computacional (*Software o Hardware*) que debe ser gestionado de forma autónoma, es necesario conocer parcial o totalmente su composición, el funcionamiento, las entradas y salidas del elemento,

límites, entre otras características, un elemento administrado puede ser un proceso, un dispositivo o incluso algo más complejo como un clúster.

- **Sensores, Actuadores y Punto de contacto**: Todo elemento que pueda ser administrado por un gestor autónomo debe estar en la capacidad de brindar información sobre sí mismo (sensores) y aportar puntos de acción que influyan directamente sobre su comportamiento o el de la acción administrada (actuadores), este punto de contacto puede ser un API-REST, una cola de trabajo, un bróker o inclusive algo tan sencillo con un log de registros.

- **Monitorear**: este proceso se encarga de recibir o compilar la información que proveen los sensores, para filtrarla, transformarla o enriquecerla de manera que se pueda producir los insumos necesarios para realizar los siguientes procesos.

- **Analizar**: en el proceso de análisis se correlaciona la información, se detectan cambios, analizan estados y en general se modela la situación actual del entorno, todo para determinar si los objetivos del sistema se están cumpliendo, en este análisis también se puede generar tendencias para la toma de acciones tempranas.

- **Planificar**: el proceso de planificar consiste en generar las acciones necesarias para acercar al sistema a su estado de equilibrio o el estado deseado, se basa en la información que provee el proceso de análisis.

- **Ejecutar:** El proceso de ejecución toma las acciones generadas por el proceso de planificación y las ejecuta, estas acciones pueden implicar eventos tan sencillos como generar un registro, enviar una alerta o acciones más complejas como actualizar un modelo de dominio o ejecutar un conjunto de comandos de autoconfiguración.

- **Conocimiento:** Es punto central de todo el ciclo MAPE-K, cada uno de los procesos anteriores acceden al conocimiento del sistema para leer o actualizar lo que el gestor autónomo sabe del elemento administrado, el conocimiento del sistema se puede representar de muchas formas, desde bases de datos relacionales, pasando por modelos de dominio, reglas de sistema, ontologías, grafos, etc.

Todo el ciclo se repite de forma periódica o en función de algún evento, de aquí que podemos tener gestores autónomos activos y pasivos, la complejidad de los gestores autónomos depende en gran medida del elemento administrado y las tareas gestionadas, el ciclo no es necesariamente lineal, en ocasiones un proceso solicita a otro que reconsidere algún evento específico o que brinde información adicional.

Mediante la implementación de componentes autónomos en elementos clave del sistema la propuesta de IBM (Horn, 2001) definió cuatro tipos de propiedades o aspectos fundamentales que pueden ser alcanzadas en un sistema autónomo, estas propiedades denominadas self-*:

- ***Self-configuration*** **o Autoconfiguración**: es la capacidad de un sistema computacional para ajustar parámetros internos de forma que pueda mantenerse en un estado inicial o para adaptarse a un estado deseado, un ejemplo de esto es la configuración de un cluster de Kubernetes que se adapta a la cantidad de instancias que debe tener una aplicación.

- ***Self-healing*** **o Auto reparación:** El sistema puede detectar, reparar o aislar fallas para maximizar la disponibilidad de sus funciones, para lograrlo es de suma importancia el manejo adecuado de errores en un sistema, un ejemplo de ello es la puesta en cuarentena o aislamiento de un sector defectuoso en un sistema de almacenamiento de datos.

- ***Self-optimization* o Auto optimización:** Los esfuerzos del sistema están orientados al uso eficiente de recursos respecto a unos lineamientos definidos, un ejemplo puede ser el escalado dinámico en algunos sistemas de computación en la nube, que permite solicitar o dar de baja recursos en función del uso de una aplicación o servicio.

- ***Self-protection* o Autoprotección:** El sistema puede anticipar, identificar y prevenir diversos tipos de amenazas con el fin de preservar su integridad y seguridad, un ejemplo sencillo de esto es un cortafuegos que detecta cuando se realizan muchas peticiones simultáneas a un mismo servicio y deniega las comunicaciones para responder ante un posible ataque de denegación de servicios.

Desde la propuesta inicial de IBM en el 2001 las propiedades self-* se han ampliado sustancialmente, hoy en día podemos encontrar un listado completo de propiedades relacionadas o asociadas a un sistema que desee poseer cierto grado de autonomía en alguno de los aspectos de su diseño o implementación. Algunos de los autores que han contribuido a definir más propiedades self-* ampliando esta lista son (Nami & Bertels, 2007) y (Poslad, 2011), algunas de las propiedades más relevantes son descritas en (Lalanda, McCann, & Diaconescu, 2013). La investigación presentada está centrada en la propiedad de autodescripción (*self-description*) también llamada autodefinición o auto representación), esta propiedad puede ser definida desde dos perspectivas, la primera es la definición y modificación de políticas de alto nivel de un sistema y la otra es la perspectiva de la comunicación, la segunda perspectiva es el aspecto desde el cual será abordada esta investigación, la definición de esta propiedad:

- ***"Self-Description o Autodescripción (Desde la perspectiva de la comunicación):** Es la capacidad de un sistema para describirse a sí mismo u otros sistemas adyacentes. La*

> *descripción de un sistema de representar un conjunto del autoconocimiento del sistema, y debe ser relevante para sus objetivos."* (Lalanda, McCann, & Diaconescu, 2013)

Eventualmente con el estudio y la búsqueda de soluciones en torno a las propiedades self-*, empezaron a aparecer soluciones autónomas a través de los años que impactaron enormemente en la tecnología de la cual disponemos hoy en día, pero si queremos comparar el nivel de autonomía de un sistema con otro no existe un estándar que permita hacerlo, sin embargo, (Jacob, Lanyon-Hogg, Nadgir, & Yassin, 2004) define 5 niveles de autonomía en los que evaluar la madurez de un sistema autónomo, estos niveles son:

- **Nivel 1 – Base:** en este nivel todos los procesos del sistema se gestionan de forma manual, aquí estaría la mayor parte de las aplicaciones de escritorio específicas, se requiere de profesional calificado para configurar, supervisar, actualizar o sustituir cualquiera de estos sistemas.

- **Nivel 2 – Gestionado**: las tecnologías de gestión y sincronización permiten generar sistemas que puedan ser controlados desde un punto en común, aquí estarían situados la mayor parte de las arquitecturas web, se automatiza la gestión de la información y es posible cambiar sistemáticamente la configuración del sistema, permite sintetizar y recopilar mejor la información en la medida que el sistema se vuelve más complejo.

- **Nivel 3 – Predictivo:** Se introducen capacidades robustas de análisis en el sistema que permiten monitorizar las situaciones que se producen en el entorno, analizarlas y tomar acciones para los posibles cursos de acción, en este nivel personal TI especializado se encarga de diseñar y desplegar los mecanismos de monitoreo y análisis, así como determinar el curso de acción a seguir, en este nivel estarían soluciones que implementen

algún mecanismo básico de gestión autónoma, dicho mecanismo se encarga básicamente de monitorear, analizar y generar acciones en función de los diferentes cambios en el contexto.

- **Nivel 4 – Adaptativo**: El sistema puede tomar decisiones autónomamente basándose en la información disponible y en los modelos de conocimiento accesibles. A medida que se generan mejores mecanismos de análisis, es posible designar parcial o completamente determinadas tareas al sistema, se requiere menos intervención por parte del personal de TI.

- **Nivel 5 – Autonómico:** El sistema se rige por políticas de alto nivel que afectan el funcionamiento de la infraestructura TI, los profesionales altamente capacitados se encargan de interactuar con las herramientas autónomas para monitorear procesos, alterar reglas de negocio u objetivos del sistema.

Es importante mencionar que los niveles de autonomía son aplicables a especificaciones de un sistema, de esta forma puede ser completamente autónomo en la tarea de mantenerse activo, pero estar en un nivel básico en lo que respecta a la forma en la cual se realizan las actualizaciones.

4.3 Arquitecturas Distribuidas

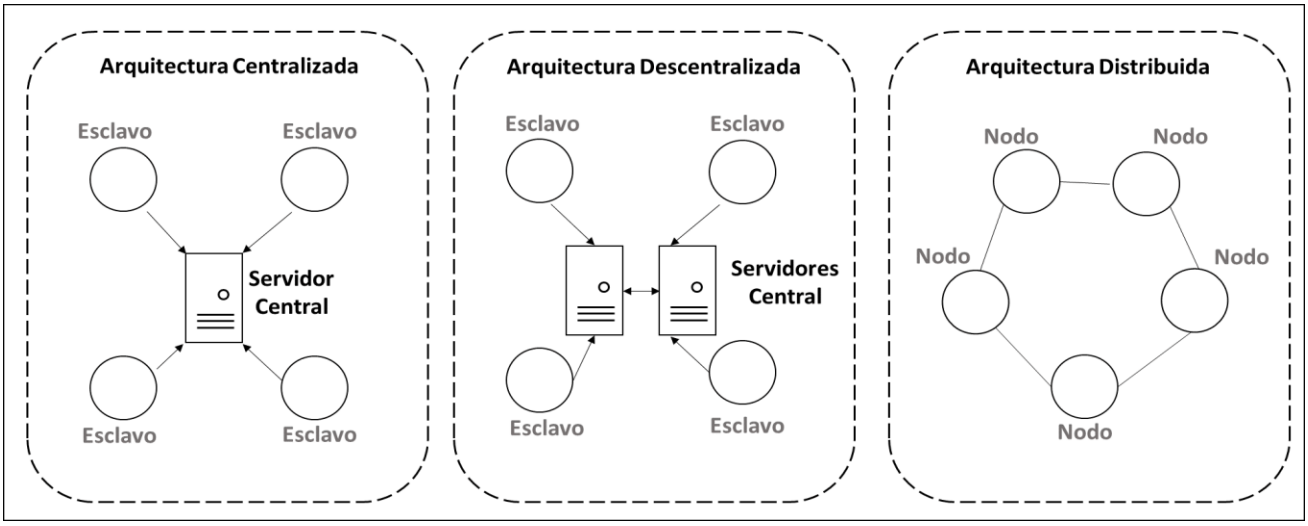
Las capacidades que tiene un equipo de cómputo para realizar tareas de procesamiento y almacenamiento de datos están limitadas por sus prestaciones, de esta forma, un solo computador siempre va a poseer un límite que puede ser fácilmente alcanzable en los sistemas modernos, por esta razón la gran mayoría de arquitecturas de altas prestaciones son arquitecturas distribuidas, el objetivo de este apartado es presentar una descripción general del concepto de arquitecturas distribuidas al lector.

Se puede definir un sistema o arquitectura distribuida a un conjunto de equipos computacionales que trabajan en conjunto para aparecer ante el usuario como un único sistema (Lopez, 2015), los sistemas distribuidos aportan características como concurrencia, modularidad, transparencia, tolerancia a fallos, escalabilidad, etc. La implementación de sistemas distribuidos proporciona grandes beneficios a un costo relativamente bajo, sin embargo, también conlleva algunos retos como el incremento en la complejidad del sistema, un mayor reto para mantener la seguridad de este y un mayor esfuerzo para el desarrollo de software distribuido.

Para entender mejor una arquitectura distribuida podemos compararla con otros modelos, en la Figura 8 se observa la representación de una arquitectura centralizada, descentralizada y distribuida, una arquitectura centralizada es el tipo de arquitectura más usado en muchas organizaciones, se tiene un servidor central donde está almacenada la información y se realiza el procesamiento de los datos, las limitantes del sistema están definidas por las capacidades de ese servidor, la verdad de la información en el sistema está registrada por ese único servidor. En una arquitectura descentralizada no se tiene un único servidor, se pueden tener varios servidores que trabajan en paralelo o sincronizados y cada uno puede tomar sus propias decisiones o ser dueño de sus propios datos, este tipo de arquitecturas toman cada vez más fuerza porque permiten tener distribuida la carga computacional, el almacenamiento y el funcionamiento en general de la plataforma. Finalmente es posible ver una representación de una arquitectura distribuida, en una arquitectura distribuida no existen servidores centrales que se encarguen de la toma de decisiones, cada nodo del sistema toma sus propias decisiones y se sincroniza con los demás nodos, estas arquitecturas son principalmente útiles cuando nos encontramos en entornos muy dinámicos, con elementos heterogéneos que entran y salen constantemente del sistema, en otras palabras es una buena opción para implementar parcial o totalmente en arquitecturas IoT.

Figura 8

Arquitectura Centralizada, Descentralizada, Distribuida



Las arquitecturas no son netamente excluyentes, es posible que para el modelo de capas de referencia IoT estudiado anteriormente (como se ha evidenciado en el estado del arte) existan arquitecturas distribuidas en cada capa, pero el sistema como un todo puede comportarse como una arquitectura descentralizada.

4.4 Modelo Conceptual o de Dominio

Un modelo de dominio o modelo conceptual puede ser definido como una representación abstracta de un conjunto de temas relacionados en torno a un contexto específico, en este modelo se describen las entidades, atributos, roles, relaciones y restricciones (Banks & Sokolowski, 2010). Los modelos de dominio son especialmente útiles cuando se quiere comprender conceptos clave de un contexto, representarlos y formar un vocabulario en torno a ello, explicar las relaciones y limitantes de sus conceptos, permiten estandarizar la visión de cómo deben o deberían ser los elementos que componen un sistema que forma parte de un dominio particular.

4.5 Grafos de Conocimiento

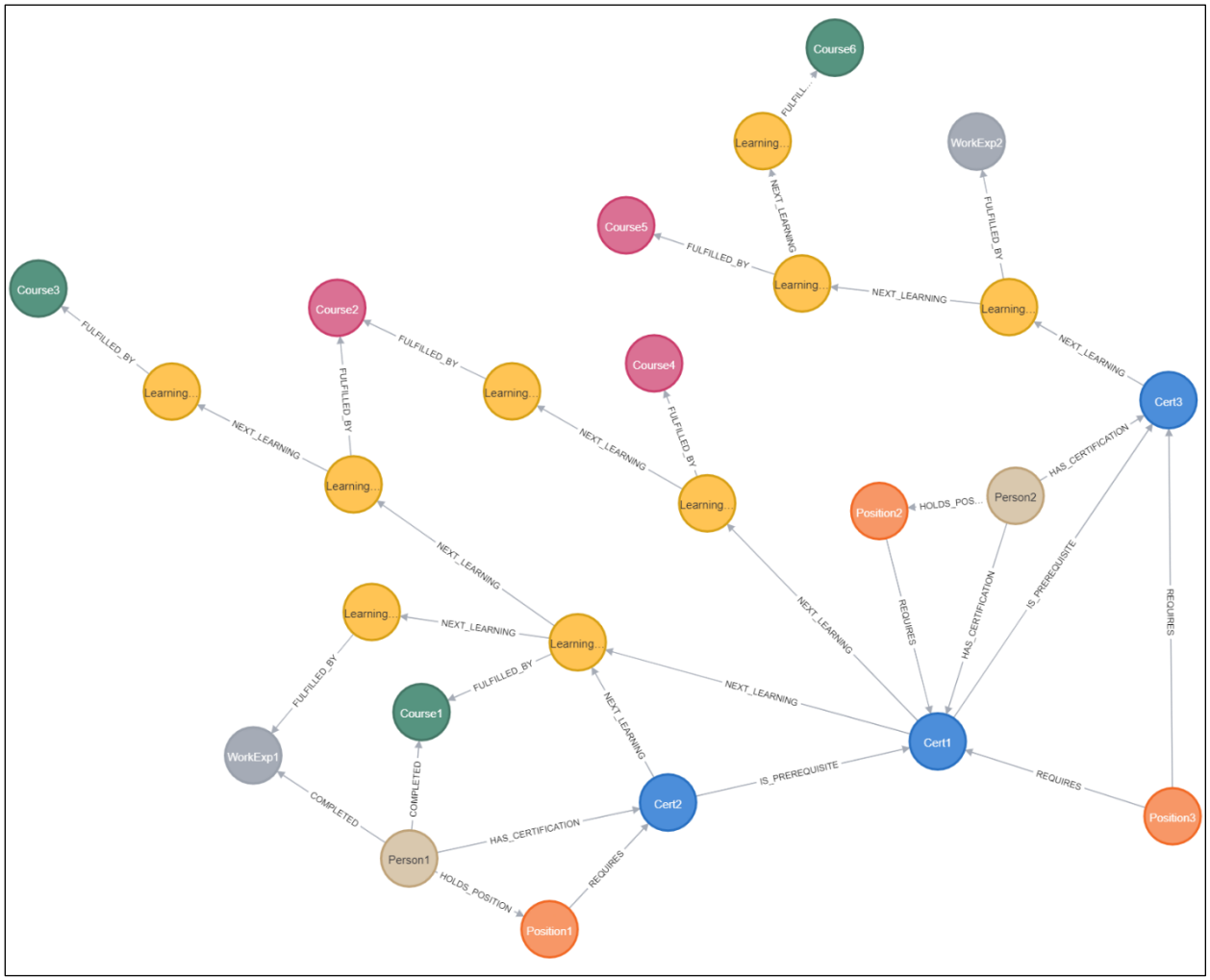
Los grafos de conocimiento son una forma de representar entidades y relaciones de un dominio específico, son de gran utilidad cuando se quiere realizar razonamiento sobre una base de conocimiento definida, al igual que los modelos de dominio, los grafos son una forma de representar los datos, hoy en día se usan ampliamente con algunas herramientas tecnológicas para obtener una implementación funcional de un modelo abstracto, de hecho, los trabajos presentados por (Smajevic & Bork, 2021) y (Chen, Jia, & Xiang, 2020) demuestran cómo es posible generar grafos de conocimiento a partir de modelos de dominio, que pueden ser implementados en sistemas de bases de datos orientados a grafos como (Neo4j Inc, 2021), estas implementaciones permiten integrar el conocimiento con otros sistemas, y realizar razonamiento sobre la base del conocimiento presentado.

En la Figura 9 se muestra el ejemplo de un grafo para modelar sistemas de gestión de aprendizaje (LMS), la idea detrás de esta representación es mapear las rutas de aprendizaje asociadas a una certificación dentro de los sistemas de gestión de aprendizaje que grandes organizaciones pueden implementar, el ejemplo de (Misquitta, 2022) busca responder algunas preguntas mediante el modelo implementado, de forma general, los nodos azules son certificaciones, los nodos amarillos elementos de aprendizaje, los rosados son cursos, los marrones personas, los naranjas son cargos y los grises experiencia laboral.

Los nodos y relaciones presentados permiten responder a preguntas como ¿Cuáles son los requisitos para una certificación? ¿Cuáles son los posibles caminos para obtener determinada certificación? ¿Qué personas de la empresa que ocupan un cargo X no están certificadas para dicho cargo? ¿Cómo empleado si busco ocupar un cargo determinado que requisitos me faltan?

Figura 9

Ejemplo de Grafo de Sistema de Gestión de Aprendizaje Implementado en Neo4j



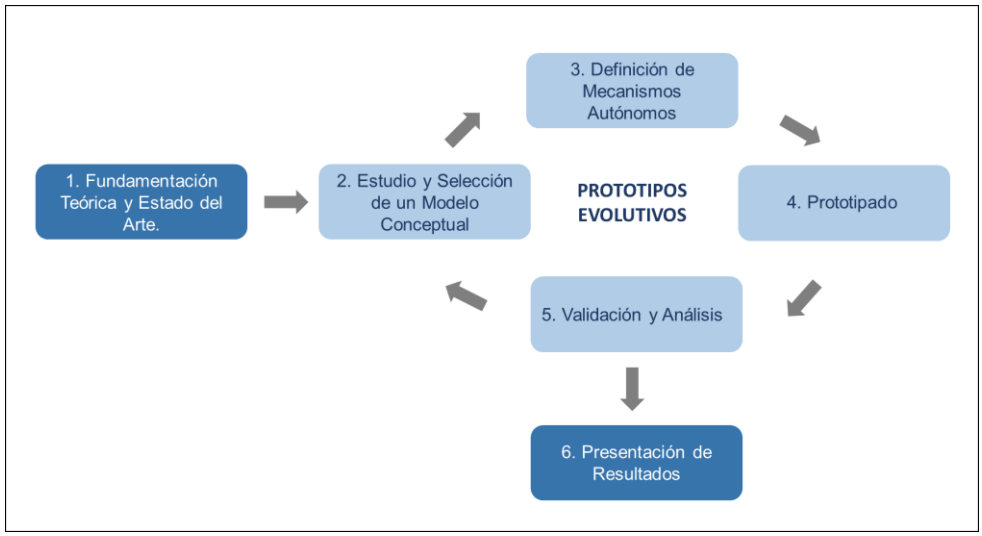
Nota: adaptado de (Misquitta, 2022)

5. Metodología de Investigación

Este apartado se dedica a presentar la metodología de trabajo que fue usada para esta investigación, el propósito de esta metodología basada en un modelo de desarrollo software de prototipos evolutivos fue avanzar en iteraciones cortas para la construcción de cada resultado propuesto, permitiendo mejorar, evaluar, agregar o descartar nuevas funcionalidades en cada iteración. La Figura 10 presenta cada fase de la metodología usada y su relación con las demás fases, posteriormente se presenta una breve descripción, así como las actividades y productos obtenidos.

Figura 10

Metodología de Investigación



5.1 Fundamentación Teórica y Estado del Arte

Esta fue la fase inicial de la metodología de investigación, en ésta el objetivo era reforzar o adquirir conocimientos teóricos relacionados con el contexto del problema estudiado, así como permitir realizar una búsqueda más profunda del estado del arte, este proceso fue transversal a

todas las demás fases ya que en la medida que se realizaban avances, también se comparaban y buscaban nuevas fuentes contra las cuales contrastar los resultados obtenidos.

Actividades

- **A1.1:** Estudio de Fundamentos Teóricos.
- **A1.2:** Construcción del Estado del Arte.

Productos

- **P1.1:** Marco Teórico de la Investigación.
- **P1.2:** Estado del Arte.

5.2 Estudio y Selección de un Modelo Conceptual

El objetivo de esta fase fue estudiar las diferentes propuestas de modelos conceptuales, compararlas y determinar similitudes entre ellas, de esta forma poder determinar a criterio del autor, cuál era la propuesta más completa para usarla. Sin embargo, se decidió proponer un modelo complementario tomando lo mejor de las propuestas estudiadas y que se adaptara mejor a la visión de arquitecturas IoT que se tiene en la investigación.

Actividades

- **A2.1:** Estudio de Propuestas de Modelos Conceptuales en Arquitecturas IoT.
- **A2.2:** Seleccionar, Adaptar o Proponer un Modelo Conceptual.

Productos

- **P2.1:** Análisis de Modelos Conceptuales Estudiados.
- **P2.2:** Modelo Conceptual de la Investigación.

5.3 Definición De Mecanismos Autónomos

Conforme al modelo conceptual que se seleccionó para la investigación, se propuso en esta fase la definición o diseño de los mecanismos autónomos encargados de monitorear los diferentes componentes software de la arquitectura para la generación autónoma de representaciones. En esta fase también fue necesario estudiar las arquitecturas de mecanismos autónomos propuestos para diferentes tareas y la forma en la cual se implementan en un sistema desde cero o en sistemas que ya existen.

Actividades

- **A3.1:** Definir y/o diseñar los de Mecanismos Autónomos para autorrepresentación de elementos software en arquitecturas IoT.

Productos

- **P3.1:** Diseño de los Mecanismos Autónomos.

5.4 Prototipado

Como parte de la validación de los mecanismos autónomos propuestos, se implementaron prototipos que permitieron añadir o descartar requerimientos según su utilidad. Estos prototipos fueron implementados en la plataforma Smart Campus que se ha desarrollado en la UIS, por el conocimiento que tenía el autor de esta y la disposición del código fuente de cada componente a modificar.

Actividades

- **A4.1:** Implementación de Prototipos.

Productos

- **P4.1:** Prototipos de los Mecanismos Autónomos.

5.5 Validación y Análisis de Resultados

Con la implementación del prototipo de los mecanismos autónomos en el sistema, fue necesario definir un diseño experimental, ejecutarlo y analizar los resultados para la siguiente iteración. El objetivo del experimento fue analizar inicialmente cómo se estaba describiendo el sistema, que fallas se presentaban y posteriormente la forma de generar esta representación autónomamente, todo esto dentro de un sistema distribuido.

Actividades

- **A5.1:** Diseñar el experimento.
- **A5.2:** Ejecución del experimento.
- **A5.3:** Análisis de resultados.

Productos

- **P5.1:** Diseño Experimental.
- **P5.1:** Resultados y Análisis Resultados.

5.6 Presentación de Resultados

En esta fase el objetivo fue la elaboración de un artículo y/o ponencia con los resultados parciales o totales de la investigación. Además, en la medida que fueron realizadas cada una de las fases anteriores se redactó el documento final de la tesis.

Actividades

- **A6.1:** Elaboración de artículo o ponencia.
- **A6.2:** Escritura del Documento Final Tesis de Investigación.

Productos

- **P6.1:** Artículo Publicado y/o Ponencia realizada.
- **P6.1:** Documento Final Tesis de Investigación.

6. Modelo Conceptual

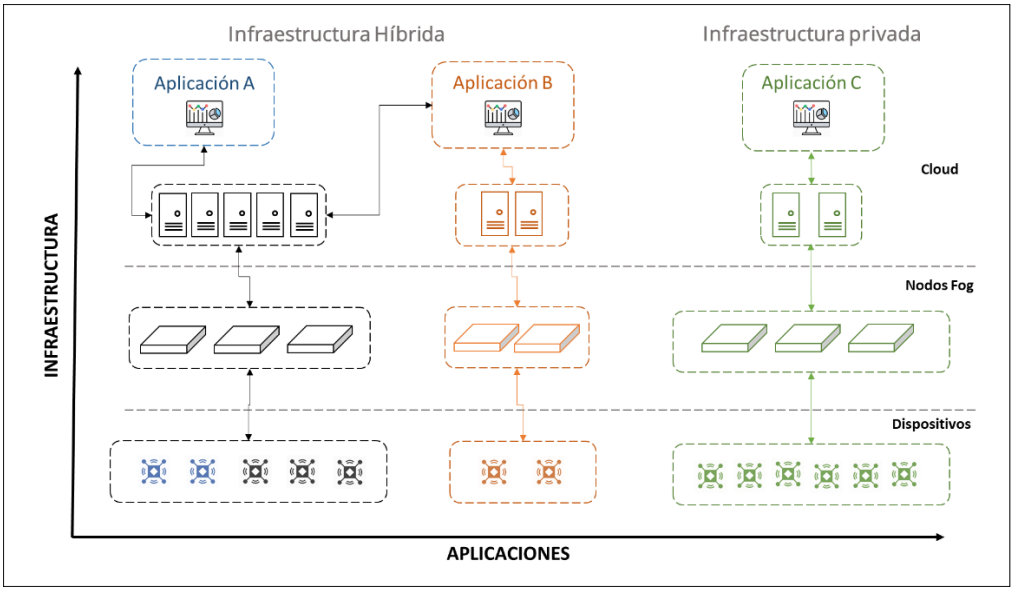
En este capítulo se presenta el modelo conceptual propuesto, partiendo de la descripción de la visión que tienen los autores de una arquitectura IoT, posteriormente se establece el modelo conceptual propuesto.

6.1 Visión De una Arquitectura IoT

Nuestra visión de una arquitectura IoT es la de un ecosistema híbrido en el que puedan existir componentes públicos, pero también componentes privados, todo ello soportado en una infraestructura distribuida que permita el despliegue de dispositivos y aplicaciones de forma independiente, así como su asociación, en la Figura 11 se puede apreciar el esquema general de nuestra visión de una arquitectura IoT, los ejes principales de esta visión son el eje de infraestructura y el eje de aplicaciones.

Figura 11

Visión de una Arquitectura IoT



La infraestructura privada permite a organizaciones desplegar sus propias aplicaciones y disponer de recursos que están destinados únicamente para el funcionamiento de dichas aplicaciones, esto implica una inversión más grande de recursos, pero garantiza la disponibilidad de la infraestructura, la flexibilidad y el rendimiento de esta, en el esquema anterior sería la aplicación C quien estaría desplegada en el ecosistema de esta forma. Por otra parte, la infraestructura híbrida permite a organizaciones o individuos desplegar sus aplicaciones y dispositivos soportados en la infraestructura pública del ecosistema, pero también permitiendo aislar parte de estos recursos a manera de infraestructura privada, en el esquema presentado la aplicación B estaría usando recursos tanto de la infraestructura pública como de su infraestructura privada, la aplicación A estaría completamente soportada en la infraestructura pública donde aporta algunos dispositivos, la idea también sería que la infraestructura pudiera cambiar de forma dinámica para ser pública o privada en función de las necesidades de las aplicaciones desplegadas, para desarrollar un poco más esta visión es necesario considerar algunas premisas:

- La arquitectura debe contar con un mecanismo de descubrimiento e identificación para cada componente del sistema (Ariza, Mendoza, Garcés, & Cardozo, 2018), este reto de reconocimiento debe ser abordado mediante la incorporación de estándares en las arquitecturas IoT y sistemas de inferencia como el presentado en (Ariza, Garcés, Cardozo, Rodríguez, & Jiménez, 2021).

- Aunque la infraestructura desplegada sea pública o privada, los elementos no deben actuar como cajas negras, por lo que deben poseer mecanismos para conocer su estructura y la de los elementos que lo componen.

- Debe existir un acuerdo común entre los conceptos y sus relaciones que sea usado por todos los actores del sistema, de forma que se facilite la comunicación y estandarización en el despliegue de nuevos dispositivos y aplicaciones.

6.2 Modelo Conceptual Propuesto

Con base en la comparativa de los modelos conceptuales encontrados en la literatura y presentados en la Tabla 2 se determinó proponer un modelo conceptual que incluya todos los conceptos relacionados en dicha comparativa y presentados en la Tabla 1, de forma que el modelo conceptual propuesto presente una visión complementaria a los modelos conceptuales encontrados como se muestra en la Tabla 4.

Tabla 4

Comparativa diferentes propuestas de modelos conceptuales

Propuesta	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
(Haller, Serbanati, Bauer, & Carrez, 2013)	NO	SI	NO	SI	NO	SI	NO	NO	SI	SI	SI
(Negash, Westerlund, Rahmani, Lijieber, & Tenhunen, 2017)	NO	SI	NO	SI	NO	SI	NO	SI	SI	SI	SI
(Costa, Pires, & Delicato, 2019)	SI	NO	NO	SI	NO	SI	NO	SI	NO	SI	SI
(Alulema, Criado, & Iribarne, 2019)	NO	SI	NO	SI	SI	NO	NO	NO	SI	SI	NO
(González, Meana, García, Jiménez, & Petearson, 2020)	SI	SI	NO	SI	NO	SI	NO	NO	SI	NO	NO
(Alfonso, Garcés, Castro, & Cabot, 2021)	SI	SI	NO	SI	NO	SI	SI	NO	SI	NO	NO
Propuesta Modelo Conceptual	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI	SI

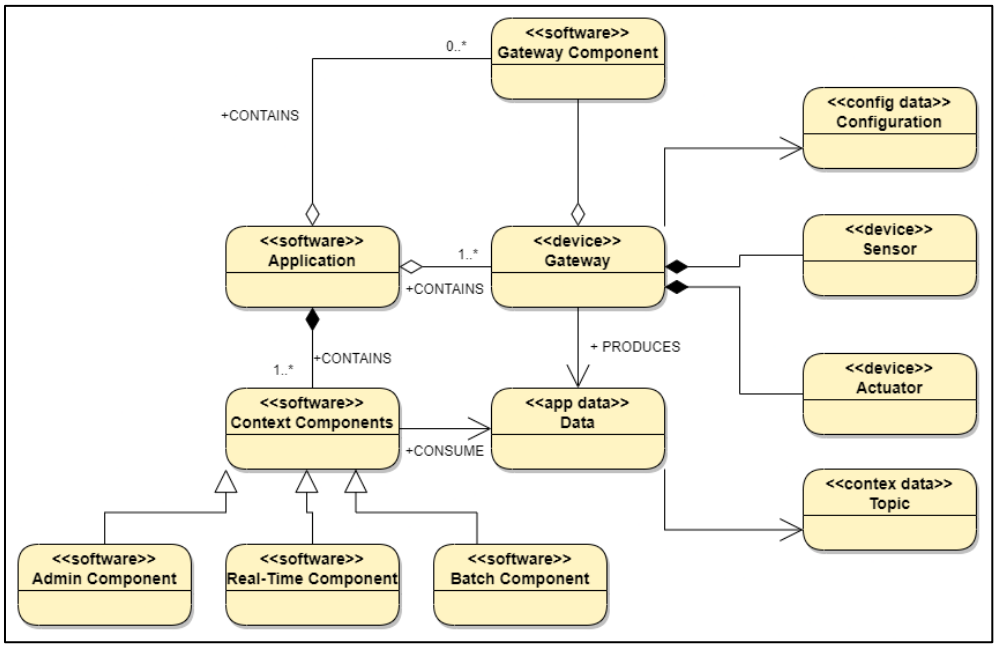
Nota: C1-Aplicación, C2-Actuador, C3-Almacenamiento, C4-Comunicación, C5-Datos, C6-Dispositivo, C7-Gateway, C8-Propiedades, C9-Sensor, C10-Servicio, C11-Recurso.

El modelo propuesto fue desarrollado en dos etapas, la primera fue la elaboración de un modelo parcial para la discusión de las relaciones entre los conceptos asociados, posteriormente se realizó la propuesta del modelo conceptual usado en el desarrollo de los demás componentes de la investigación.

La primera versión del modelo conceptual se presenta en la Figura 12 y fue una de las premisas del trabajo presentado en (Jiménez, Cárcamo, & Pedraza, 2020), para su elaboración se tuvieron en cuenta principalmente los conceptos de Aplicación, Gateway, Datos, Sensores y Actuadores, los servicios se presentaban como componentes del contexto y los Gateway poseen componentes software tales como bases de datos, brokers, aplicaciones, etc.

Figura 12

Modelo Conceptual Versión 1



Aunque el modelo ayudó a definir mejor la visión de los autores, también reveló algunas desventajas presentes:

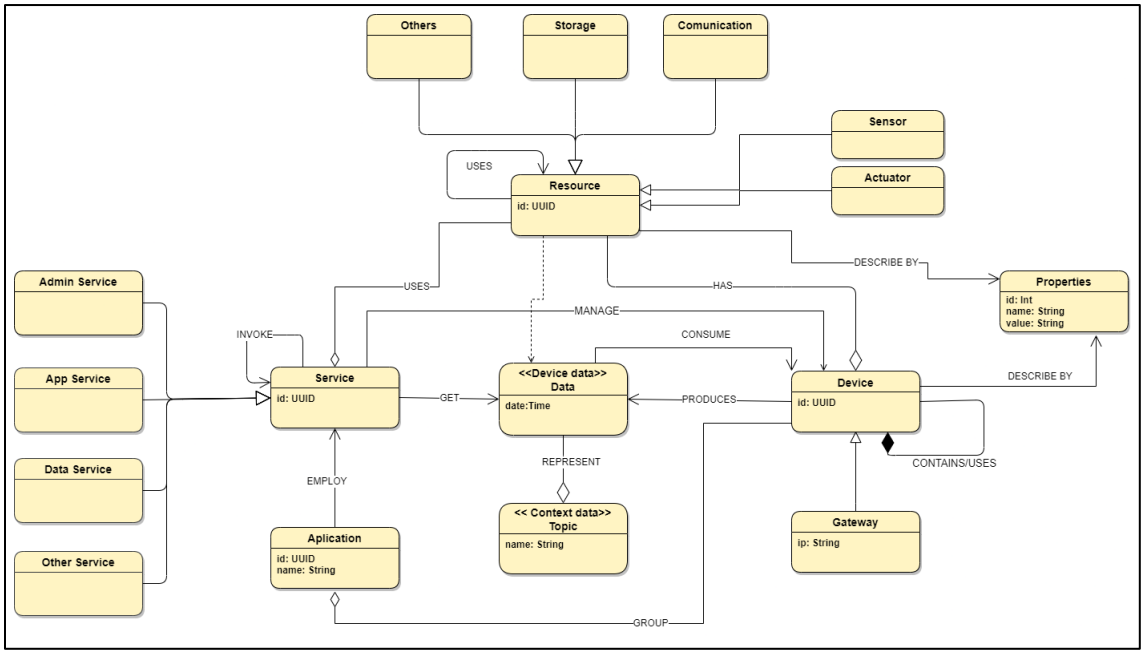
- El modelo no se adapta muy bien para elementos en arquitecturas distribuidas.
- Los dispositivos son sensores y actuadores que obligatoriamente requieren un Gateway para integrarse en la arquitectura.
- No se puede mapear correctamente elementos de la nube como bases de datos, brokers y servicios en general.
- Las aplicaciones son las que definen los servicios.

Posteriormente se realizó el diseño de una nueva versión del modelo conceptual, se presenta en la Figura 13, este modelo contempla nuevas formas de representar los componentes

software en una arquitectura IoT, se integra el concepto de recurso, servicio y propiedades en el modelo.

Figura 13

Modelo Conceptual Versión 2



En el modelo propuesto se observa que los servicios, aplicaciones, dispositivos y recursos tienen un id como UUID, esto viene de Universal Unique Identifier o identificador único universal.

A continuación, se resumen los componentes descritos en la versión 2 del modelo conceptual, teniendo en cuenta que está orientado principalmente a la descripción de componentes software:

Recurso: es un elemento abstracto que permite enlazar componentes a los cuales se puede acceder, y que brindan algunas prestaciones a otros elementos de la arquitectura, inclusive siendo usados por otros recursos, permite representar sensores, actuadores, sistemas de almacenamiento como bases de datos, elementos de comunicación como brokers y está abierto a la posibilidad de

integrar otros tipos de elementos, como por ejemplo un recurso de logs del sistema o una API externa.

Dispositivos: Los dispositivos tienen acceso a recursos y representan una agrupación de estos, permiten modelar componentes sencillos y complejos, un dispositivo puede contener o ser la composición de otros dispositivos.

Gateway: Los gateways o puertas de enlace se representan en el modelo como una clase particular de dispositivos, poseen capacidades de cómputo, almacenamiento y comunicación, pero pueden ser vistas como dispositivos que gestionan y contienen otros dispositivos, generalmente hacen parte de la capa edge computing o de la capa fog computing.

Propiedades: Tanto dispositivos como recursos poseen un conjunto de propiedades para describir sus características, la naturaleza de estas propiedades depende de la complejidad del dispositivo y el formato de implementación, se representan con un formato clave-valor el valor puede ser sencillo cómo un valor único hasta representaciones más robustas como un JSON.

Servicio: Un servicio es un componente que agrupa elementos software orientados a cumplir con un objetivo particular, un ejemplo de ello sería un servicio de administración que se encarga del registro y seguimiento de la información de componentes de la plataforma, otro ejemplo sería un servicio de mensajes que se encarga del procesamiento y almacenamiento de series temporales y datos que en general producen los dispositivos, los servicios hacen uso de recursos como bases de datos, elementos de comunicación como *brokers*, colas de trabajo, etc.

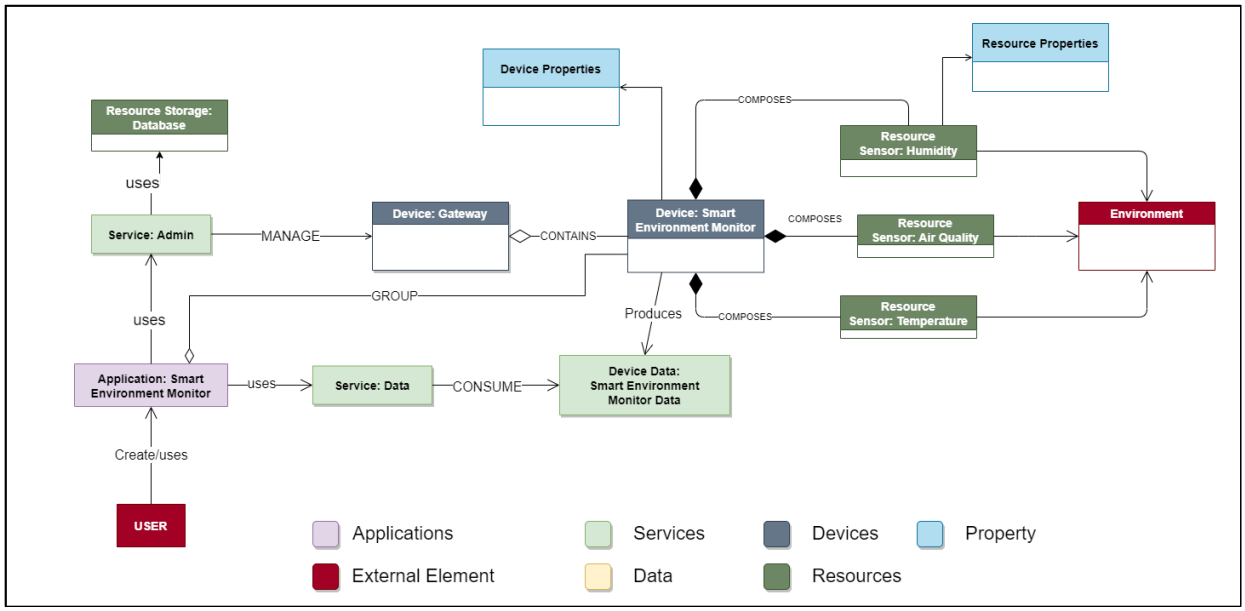
Datos: los datos son producidos por los dispositivos, estos representan un tópico y son consumidos por los servicios para ser puestos a disposición del sistema, la naturaleza y rigidez de estos datos depende de la implementación y escala de la arquitectura.

Aplicación: Una aplicación es un proceso asociado a la lógica de negocios de un contexto, pueden agrupar varios dispositivos que pueden ser compartidos con otras aplicaciones y emplea los servicios de la arquitectura para el acceso a componentes de esta.

Para comprender mejor la utilidad de un modelo conceptual se presentan dos ejemplos de configuraciones IoT y cómo pueden ser representadas utilizando el modelo conceptual propuesto, el primero de ellos es una estación de monitoreo ambiental que se muestra en la Figura 14.

Figura 14

Ejemplo de estación de monitoreo ambiental



En este contexto una estación de monitoreo ambiental es un dispositivo compuesto por tres recursos, los tres sensores son de humedad, calidad de aire y temperatura respectivamente, tanto los recursos como el dispositivo tienen propiedades como referencia, rangos de funcionamiento, identificador, fecha de creación en el sistema, entre otras, luego el dispositivo de monitoreo ambiental produce datos que son consumidos por el servicio de manejo de datos y además es

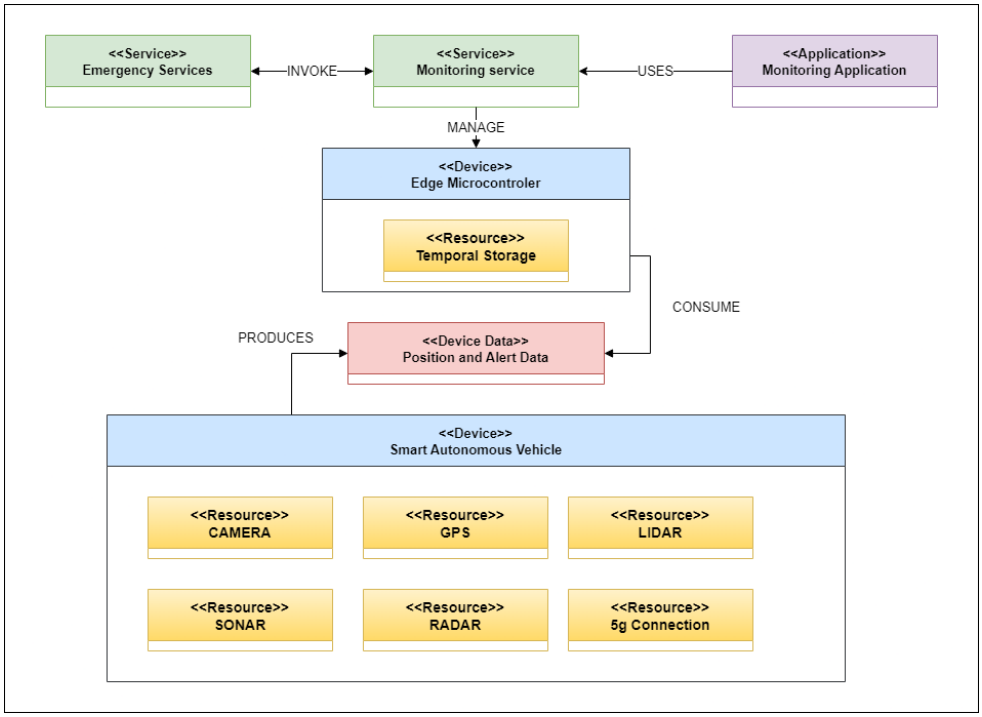
gestionado por otro dispositivo que es un gateway, el gateway está sincronizado y recibe lineamientos del servicio de administración del sistema.

El servicio de administración del sistema hace uso de un recurso de almacenamiento que es una base de datos, finalmente el acceso a la información y a la administración de los dispositivos la hace el usuario a través de su aplicación de monitoreo ambiental, esta representación permite comprender la composición de la estación de monitoreo ambiental y como es la arquitectura de la aplicación desplegada, sin embargo, se pueden añadir elementos adicionales como los recursos de comunicación usados como un broker de mensajería.

En la Figura 15 se presenta un ejemplo de modelado para un vehículo autónomo, los vehículos autónomos vienen equipados con gran cantidad de sensores, en el ejemplo se representan algunos como las cámaras, el GPS, el sistema Lidar, Sonar, Radar y un recurso para la conexión 5g. El coche autónomo genera datos de posicionamiento y alertas que consume un nodo Edge encargado de almacenar y procesar temporalmente la información del vehículo, así como transferir la información preprocesada al servicio de monitoreo, el cual se encarga de invocar al servicio de emergencia en caso de ser necesario, la aplicación de monitoreo hace uso del servicio de monitoreo para leer datos de posicionamiento de diferentes vehículos.

Figura 15

Ejemplo de vehículo Autónomo



El modelo conceptual propuesto permite modelar diferentes configuraciones de arquitecturas IoT desde una perspectiva abstracta para comprender la composición y las relaciones de los componentes software y hardware, este modelo será usado como representación base para la construcción de los componentes autónomos para la autodescripción y la posterior implementación del prototipo de validación.

7. Diseño de Mecanismos Autónomos

El término “autodescripción o autorrepresentación” se refiere a la capacidad de un sistema para describirse a sí mismo en función de sus componentes y las relaciones de estos, así como los sistemas cercanos, el objetivo de esta investigación está centrado en generar una representación con base en los componentes software de una arquitectura IoT distribuida, usando el conjunto de conceptos y relaciones representado en el modelo de dominio presentado en el capítulo anterior, a continuación, se describe el proceso de autodescripción, luego se aborda la definición de los adaptadores de autodescripción, se presenta el diseño para el mecanismo de autodescripción y finalmente se presenta la propuesta para adaptar dichos mecanismos en la arquitectura IoT Smart Campus UIS.

7.1 Proceso de Autodescripción

Para agregar capacidades autonómicas en una arquitectura IoT se deben establecer algunas condiciones antes del estudio a profundidad del sistema, para esta investigación se parte de tres premisas:

I. La arquitectura IoT se encuentra en constante desarrollo y sus componentes no son cajas negras, por lo tanto, es posible acceder al código y adaptarlo.

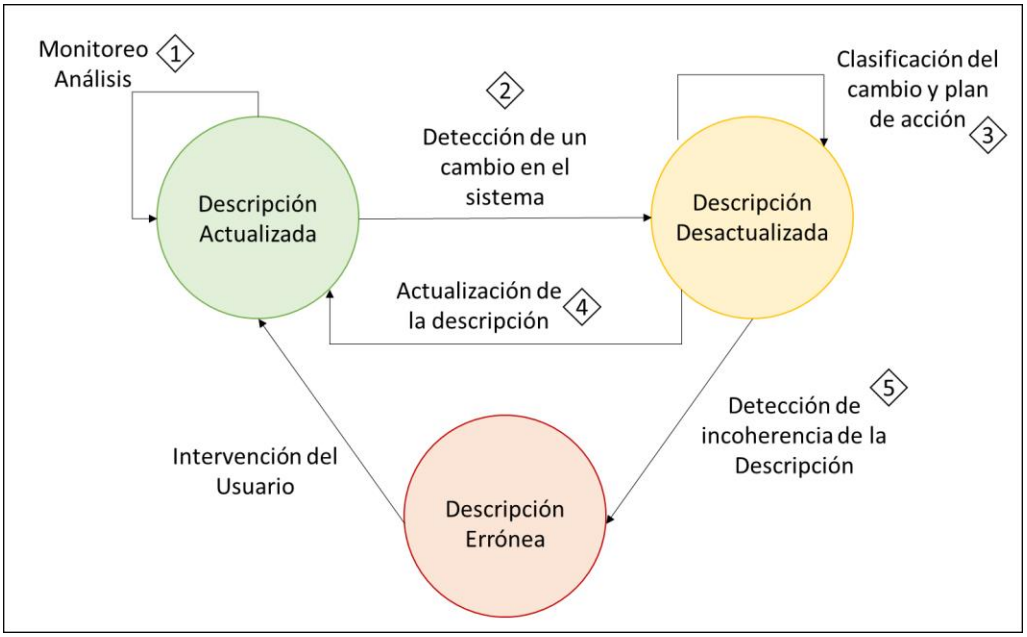
II. Es posible hacer pruebas en un entorno controlado de la arquitectura y esta no posee una alta demanda en dicho entorno que pueda afectar la calidad del servicio o los resultados de las pruebas.

III. Los componentes del sistema cuentan con algún mecanismo de comunicación por medio del cual se pueda obtener y enviar información, en caso de no existir es posible adaptarlos para implementar dichos mecanismos.

Establecidas las condiciones del sistema es importante entender las acciones que pueden generar un cambio en la descripción de este, en la Figura 16 podemos observar el diagrama de los posibles estados que puede tener la representación del sistema y las transiciones entre estados.

Figura 16

Diagrama de estados para la autodescripción del sistema y transiciones



En el estado estable de descripción actualizada el mecanismo de autodescripción conoce el estado actual del sistema y la representación generada es coherente, 1) en el estado estable se monitorea y analiza las novedades del sistema, 2) se detecta un cambio en el sistema, por ejemplo ingresa un nuevo componente en la arquitectura, lo que implica que la descripción ya se encuentra desactualizada y no coincide con la realidad, esto nos lleva al estado de descripción desactualizada,

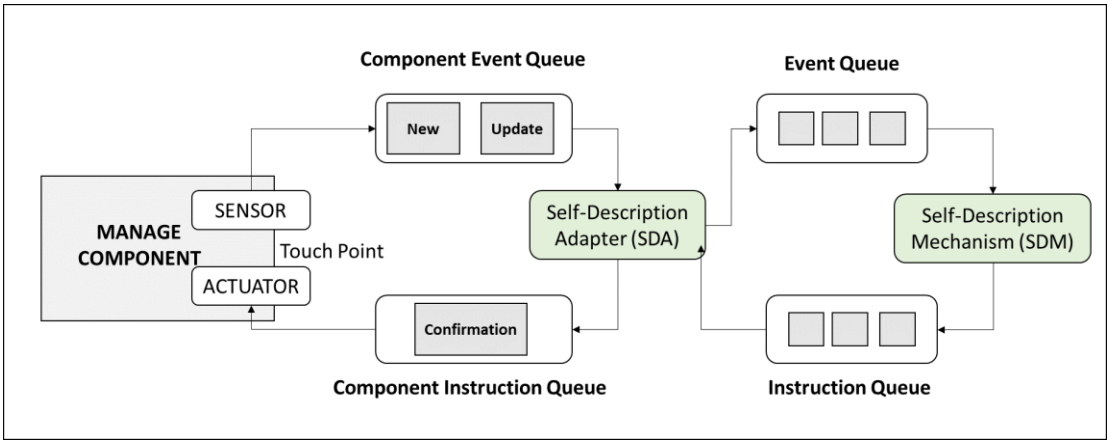
por lo tanto es necesario 3) clasificar el cambio para determinar qué acciones se requieren sobre la representación, por ejemplo, insertar la información del nuevo componente y qué relación tiene con los componentes existentes, una vez determinado el plan de acción intenta ejectuar, esta acción puede desencadenar en dos resultados 4) la representación del sistema es actualizada y ahora es coherente con el estado actual del sistema, eso nos devuelve al estado estable, o 5) se detecta una incoherencia en la representación, por ejemplo, borrar un componente que no existe, en este caso se determina que la representación es errónea o incoherente por lo tanto se requiere de la intervención del usuario para actualizar o rehacer la representación del sistema.

7.2 Adaptadores de Autodescripción

Los adaptadores de autodescripción se encargan de procesar los datos generados por los sensores del componente gestionado, y también de procesar instrucciones o confirmaciones dadas por el mecanismo de autodescripción, la idea del diseño propuesto es generar el menor impacto posible en la carga de trabajo del componente gestionado. Debido a la escala presente en arquitecturas IoT, trabajar los eventos que surgen de forma síncrona se hace una tarea compleja que demanda muchos recursos, por esta razón se decidió procesar los eventos de manera asíncrona con colas de trabajo secuenciales, en cada cola se respeta el orden de llegada de manera que el primero en llegar debería ser el primero en salir, en la figura 17 se presenta el flujo de trabajo para la generación de eventos y recepción de instrucciones en un componente gestionado.

Figura 17

Adaptadores de Autodescripción



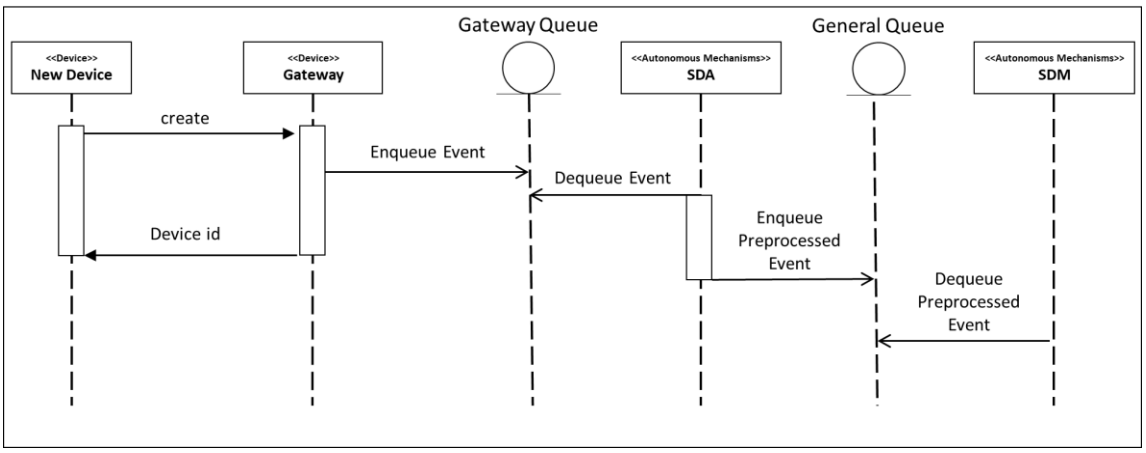
La idea de los eventos es notificar de un cambio en el sistema para analizarlo y determinar si implica una actualización en la representación del mismo, cada componente gestionado puede tener exclusivamente o compartir las colas de generación de eventos y recepción de instrucciones, el mecanismo marcado como SDA (*Self-Description Adapter*) o adaptador de autodescripción se encarga de pre procesar la información y asignarle un formato específico, de forma que pueda ser enviada al mecanismo de autodescripción mediante colas generales que son usadas por todos los SDA, también se encarga de recibir instrucciones, procesarlas y enviarlas al componente gestionado según corresponda.

Los sensores pueden ser pequeños fragmentos de código que se ejecutan con acciones específicas en el componente gestionado, por ejemplo, si el componente gestionado es un Gateway que se encarga de administrar dispositivos, y recibe la solicitud para registrar un nuevo dispositivo, el comportamiento que debería tener el sistema está ilustrado en la Figura 18, un dispositivo nuevo solicita ser creado en el sistema, el Gateway se encarga de procesar la solicitud, si el dispositivo tiene permisos y se crea en el sistema se envía el evento a la cola para ser procesado por el SDA,

el Gateway responde al nuevo dispositivo con el ID asignado en el sistema, de forma asíncrona el SDA procesa el evento y lo envía a la cola general para ser procesado posteriormente por mecanismo de autodescripción que decidirá si dicho evento implica una actualización en la representación del sistema.

Figura 18

Diagrama de Secuencia Nuevo Dispositivo

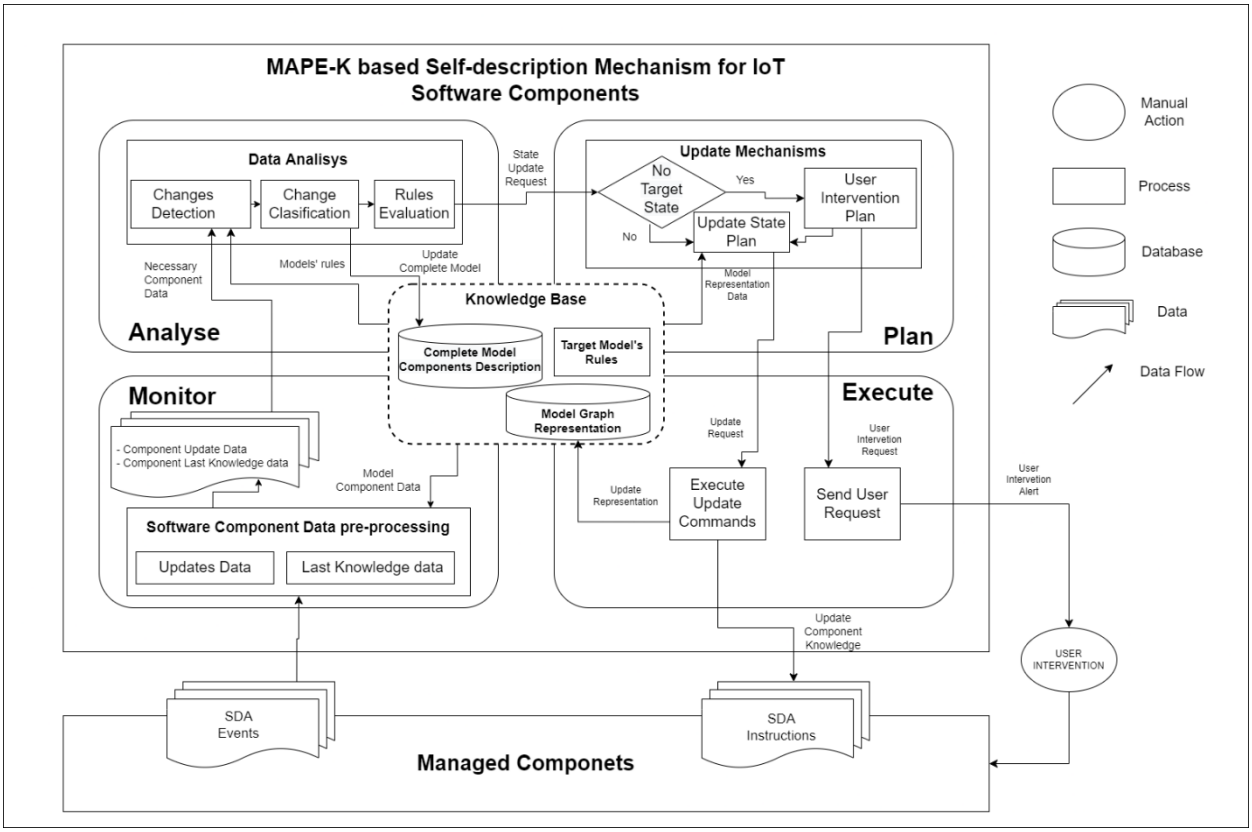


7.3 Mecanismo de Autodescripción

El objetivo principal del mecanismo de autodescripción es procesar los diferentes eventos generados en la arquitectura IoT y determinar las acciones necesarias para mantener actualizada la representación de ésta, en caso de encontrar una incoherencia genera alarmas para la intervención del usuario, el diseño del mecanismo estaba basado en el ciclo MAPE-K y posee dos formas de representación del modelo conceptual como base del conocimiento, en la Figura 19 se puede apreciar del diseño de este mecanismo que será explicado a detalle a continuación:

Figura 19

Mecanismo de Autodescripción



Base de conocimiento: es la parte central del mecanismo de autodescripción, representa el modelo conceptual que se almacena en diferentes fuentes de información, el diseño propuesto considera dos representaciones del modelo, una de ellas es una descripción completa de los componentes software de la arquitectura IoT, esto quiere decir que se almacenan todas las propiedades, atributos e incluso historia proporcionada por los diferentes mecanismos de transferencia de datos, es el repositorio principal de información, luego está la representación de la arquitectura como un grafo de conocimiento, esta permite modelar mejor las relaciones entre los componentes y realizar análisis sobre elementos clave, almacena solamente las propiedades clave y el estado actual en el sistema, finalmente existen un conjunto de reglas que son aplicadas

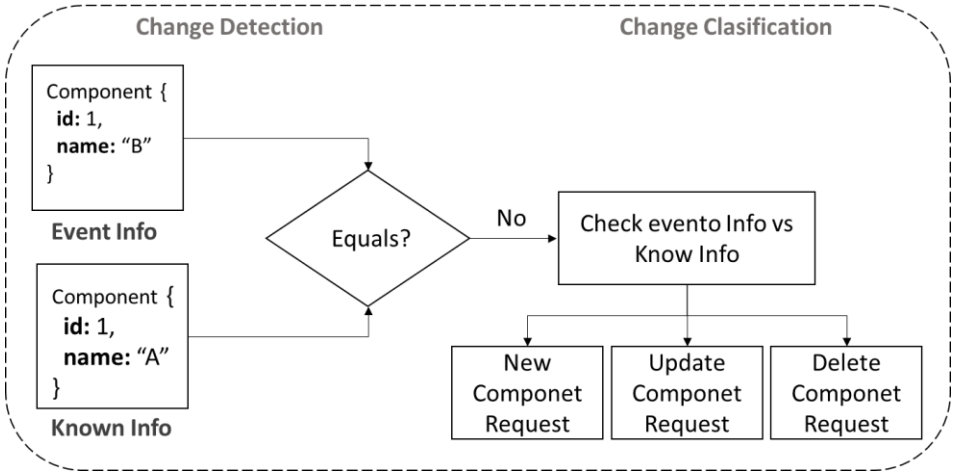
en el análisis de la arquitectura, estas reglas de alto nivel son definidas por el administrador del sistema y permiten generar advertencias cuando no se cumplen, una regla por ejemplo puede ser que exista un número mínimo de dispositivos para una aplicación desplegada.

Monitoreo: Es el primer proceso del ciclo MAPE-K, se encarga de leer los eventos generados que se encuentran en la cola de eventos, pre procesarlos y complementar la información proporcionada con la última información conocida de cada componente, también se encarga de filtrar eventos que no aportan conocimiento nuevo de la arquitectura, su objetivo principal es brindar los insumos necesarios para el proceso de análisis, el rol en la base de conocimiento de la arquitectura IoT es solamente de lectura.

Análisis: es uno de los procesos fundamentales del mecanismo de autodescripción, se encarga de detectar cambios en el sistema, clasificarlos y verificar las reglas de alto nivel establecidas por el administrador, para esto analiza la nueva información proporcionada de cada componente en un evento dado y compara la última información conocida como se presenta en la Figura 20, de esta manera se detectan componentes nuevos, cambios en componentes existentes o componentes que se eliminaron, en caso de generarse un evento donde la novedad reportada sobre la información del componente no representa cambio alguno se ignora dicho evento, de la misma forma se reportan novedades sobre las relaciones de los componentes, en el análisis también se actualiza la representación completa de la arquitectura IoT y genera solicitudes para el proceso de planificación.

Figura 20

Análisis de Componentes



Planificación: una vez clasificados los cambios y evaluado las reglas de alto nivel en el análisis, el proceso de planificación se encarga de determinar las acciones necesarias con base en las solicitudes que reciba, estas acciones están principalmente orientadas a actualizar la representación de la arquitectura IoT de manera que esté sincronizada con el estado real del sistema, también evalúa la necesidad de generar alertas para intervención del usuario.

Ejecución: se encarga de ejecutar el plan de acción determinado, actualiza el grafo de conocimiento de la arquitectura IoT y envía instrucciones a los componentes gestionados, además genera notificaciones para el usuario.

Con el diseño de los mecanismos autónomos realizado se procedió a realizar la implementación de un prototipo en la plataforma Smart Campus UIS que se presenta en el capítulo 8.

8. Validación

Para la validación del diseño propuesto se realizó la implementación de un prototipo de los mecanismo de autodescripción en la plataforma Smart Campus UIS, se seleccionó esta plataforma por la experiencia y el conocimiento que se tiene del desarrollo realizado y el funcionamiento de cada uno de sus componentes, a continuación, se presenta la arquitectura de la plataforma Smart Campus UIS, luego las tecnologías seleccionadas en la implementación del prototipo de los mecanismos autónomos, posteriormente se describe diseño experimental propuesto y finalmente se presenta el análisis de resultados, el código de todos los mecanismos implementados y los componentes de la arquitectura IoT Smart Campus UIS se encuentran en el repositorio de código del proyecto (Jiménez & Pedraza, 2022).

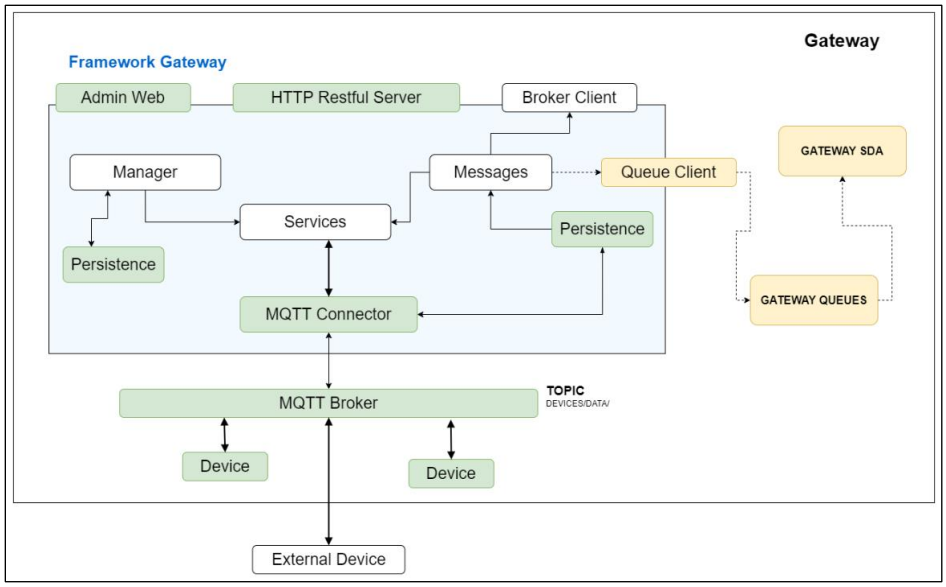
8.1 Arquitectura Smart Campus UIS

La arquitectura IoT seleccionada tiene componentes implementados que permiten la integración de los recursos, dispositivos, servicios y aplicaciones, el primer componente es un framework Gateway que se despliega en tarjetas como una Rasberry Pi y permite la gestión de dispositivos así como el envió de mensajes, en la Figura 21 se presenta el esquema de este componente, consta de un bróker MQTT interno por el cual los dispositivos realizan él envió de información, estos dispositivos pueden estar conectados físicamente en el Gateway o la forma más común es que se encuentren desplegados de forma externa y envíen datos a través de la red. Internamente el Gateway se encarga de administrar los dispositivos, proveer los servicios asociados y generar la transmisión de datos por el módulo de mensajes, aprovechando este módulo se programó el envio de cambios detectados por el Gateway a la cola de eventos para ser procesado por el SDA, todo esto desplegado dentro del Gateway que tiene la capacidad de gestionar varios

dispositivos de un mismo entorno, por lo tanto, es el elemento ideal para centralizar parcialmente los cambios generados, los componentes amarillos fueron implementados y hacen parte de los mecanismos de autodescripción.

Figura 21

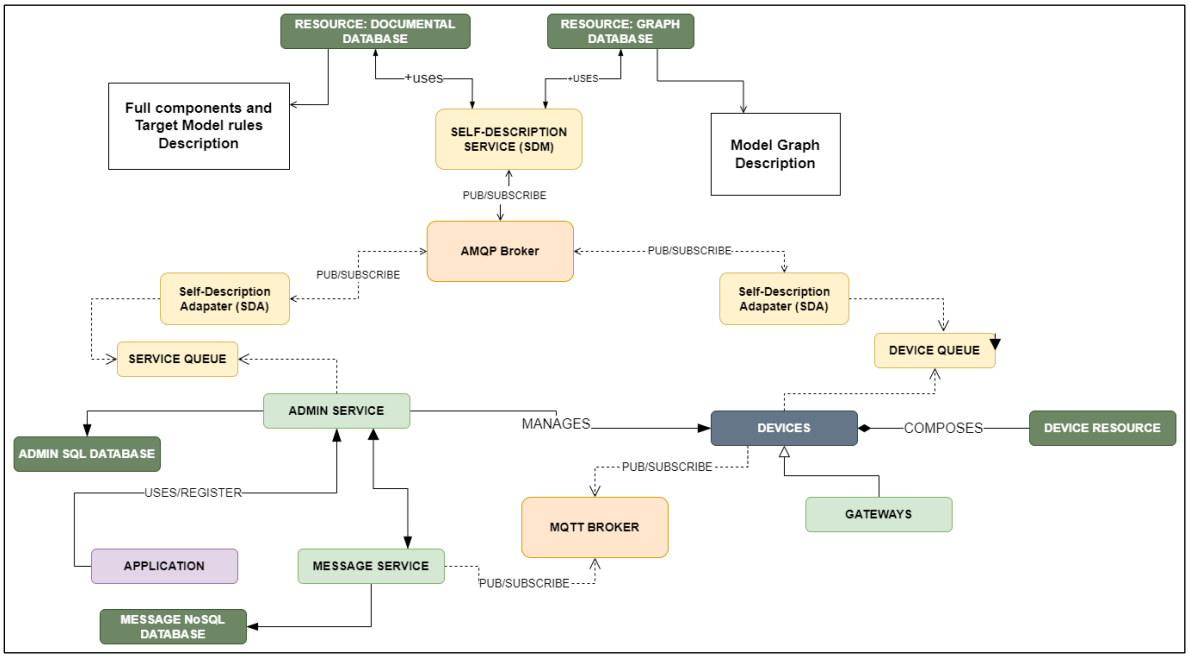
Framework Gateway



La arquitectura completa del Smart Campus UIS se presenta en la Figura 22, esta es una arquitectura que contiene dos servicios, el primero es el servicio de administración que se encarga del registro y gestión de todos los elementos del sistema, hace uso de una base de datos relacional, luego el servicio de mensajería que se encarga de la recepción de datos o mensajes enviados por los dispositivos y su almacenamiento, las aplicaciones se registran directamente con el servicio de administración, el servicio de administración también gestiona el registro y estado de los demás servicios del sistema, por lo que conoce el estado del servicio de mensajería, la arquitectura hace uso de un bróker MQTT para el envío y recepción de datos recopilados por los dispositivos.

Figura 22

Arquitectura Smart Campus UIS



Los elementos que aparecen en amarillo en el diagrama de la arquitectura Smart Campus UIS, son los mecanismos propuestos para la autodescripción, parte de las colas de mensajes que usan los diferentes componentes de la arquitectura para reportar novedades, el servicio de administración tiene su propio SDA y cada dispositivo Gateway que se despliega maneja igualmente su propio SDA, todos los SDA usan un bróker AMQP para el envio de novedades hacia el SDM, este a su vez hace uso de una base de datos documental y una base de datos orientada a grafos para las diferentes representaciones de la arquitectura con base en el modelo conceptual propuesto.

8.2 Prototipo Smart Campus UIS

Cómo se observa en la Figura 23 el prototipo de los mecanismos autónomos para la autodescripción fue implementado sobre la arquitectura existente de la plataforma Smart Campus UIS, los componentes software existentes se describen a continuación:

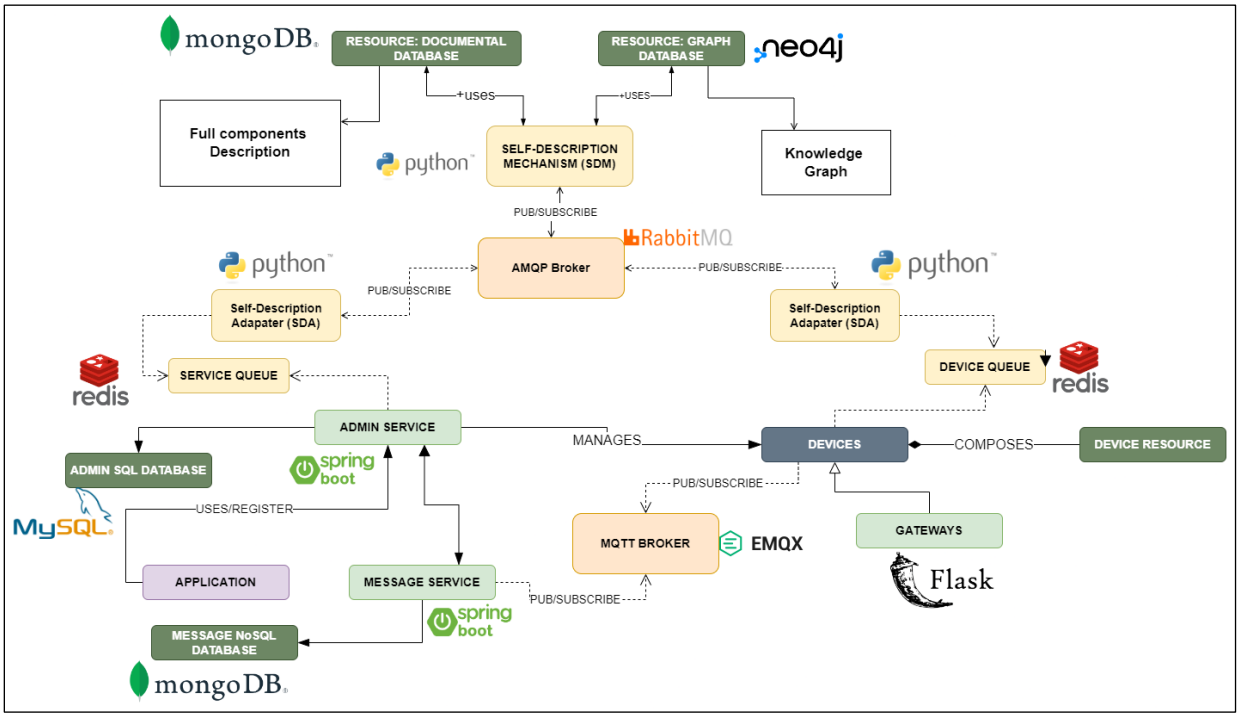
- **Gateways**: Cómo se muestra en la Figura 21 los gateways son componentes que tienen desplegado un framework y son parte fundamental de la arquitectura de Smart Campus, se encargan de la gestión de dispositivos y pueden funcionar solos o gestionados por el servicio de administración, la tecnología en la cual está implementado el Gateway es Python con la librería de desarrollo Flask.

- **Servicios**: La plataforma está conformada en la actualidad por dos servicios, un servicio de mensajes y uno de administración que están implementados en Java con el framework Spring Boot, se sincronizan con los dispositivos por HTTP y el intercambio de mensajes se hace a través de MQTT, el servicio de administración hace uso de la base de datos MySQL y el servicio de mensajería almacena los datos generados por los dispositivos en la base de datos documental Mongo DB.

- **Aplicaciones**: Las aplicaciones son elementos que implementa y registra el usuario de la plataforma, una vez registrada una aplicación puede gestionar y/o conocer información de dispositivos a través de una API-REST que expone el servicio de administración.

Figura 23

Prototipo Smart Campus UIS.



Los mecanismos de autodescripción se implementaron usando las siguientes tecnologías:

- **Colas**: Las colas de envío y recepción de datos para los componentes software fueron implementadas haciendo uso de Redis, esta tecnología dispone de un mecanismo para la creación y acceso a colas de datos, principalmente la elección de esta tecnología se debió al bajo consumo de memoria que se registra en los dispositivos, luego las colas de comunicación entre los SDA y el SDM se implementaron en RabbitMQ que permite una comunicación asíncrona distribuida y que puede escalar eficientemente.

- **Self-Description Adapater (SDA)**: Los adaptadores de autodescripción fueron implementados en el lenguaje de programación Python, haciendo uso de librerías que

permitan la comunicación con Redis y con RabbitMQ, los adaptadores revisan, procesan y vacían la cola de datos cada segundo.

- **Self-Description Mechanism (SDM):** El mecanismo de autodescripción fue implementado en el lenguaje de programación Python, se suscribe al bróker RabbitMQ para recibir las novedades del sistema, para la implementación de la representación completa del sistema se usó la base de datos documental MongoDB, esta base de datos usa JSON como formato de representación de la información. Para la generación del grafo de conocimiento se implementó la base de datos orientada a grafos más usada en el Mundo Neo4j, que además de ofrecer varias herramientas de análisis de grafos, también expone una interfaz web que facilita la gestión y consulta de los datos.

8.3 Desarrollo Experimental

En este apartado se presentan los objetivos propuestos para el experimento, las métricas usadas, el software y hardware empleado. Los objetivos propuestos para el desarrollo experimental que valide los mecanismos autónomos propuestos son:

- Medir el tiempo de respuesta entre el reporte de una novedad y la actualización de la representación del sistema y/o generación de alertas.
- Evaluar la coherencia entre la representación generada por los mecanismos autónomos y el estado real del sistema.
- Validar el rendimiento de los mecanismos autónomos ante el estrés de determinada cantidad de solicitudes y/o novedades reportadas.

- Evaluar la capacidad de resiliencia del sistema ente la falla de los componentes propuestos.

Para la evaluación de los objetivos propuestos para el experimento se definieron 5 métricas presentadas en la Tabla 5 para la oportunidad, precisión, rendimiento y resiliencia de los mecanismos implementados, para facilitar la evaluación de estas métricas se generaron sistemas de logs, además las solicitudes y respuestas fueron evaluadas desde un mismo dispositivo.

Tabla 5

Métricas de evaluación

#	CARACTERISTICA	MÉTRICA	DEFINICIÓN	EJEMPLO
1	Oportunidad (Timeliness)	Retraso de la sincronización	Tiempo promedio entre la generación de una novedad y la actualización de la representación y/o generación de alerta.	2,2 Seg
2	Precisión (Accuracy)	Tasa de pérdida	El número de novedades reportadas que se perdieron, no fueron procesadas o se procesaron mal.	5/50 (10%)
3	Rendimiento (Throughput)	Tasa de procesamiento	Cantidad de novedades exitosas evaluadas por segundo	10 x seg
4		Tasa de coherencia	Cantidad de elementos bien representados sobre el total de elementos existentes en la plataforma.	9/10 (90%)
5	Resiliencia (Resilience)	Tasa de Resiliencia	Cantidad de componentes que se bloquean o dejan de funcionar ante la falla de un componente del sistema, sobre el total de elementos existentes en la plataforma.	2/10 (20%)

El hardware usado para el experimento se resume en la Tabla 6, para las pruebas realizadas se usaron tres dispositivos Raspberry pi 4, dos de ellas con las mismas capacidades y una con menor memoria RAM, también se usaron dos equipos de cómputo para el despliegue de los servicios y el mecanismo de auto descripción respectivamente, finalmente para la comunicación se creó una red local con comunicación Ethernet y WIFI.

Tabla 6

Hardware para pruebas

#	Nombre	REFERENCIA	CPU	RAM	DISCO DURO	SO
1	Gateway 1	Raspberry Pi 4 Model B 4 Gb RAM	64-BIT Quad-core Cortex-A72	4 GB LPDDR4	32 GB Kingston MicroSD	Raspberry Pi OS
2	Gateway 2	Raspberry Pi 4 Model B 8 Gb RAM	64-BIT Quad-core Cortex-A72	8 GB LPDDR4	32 GB Kingston MicroSD	Raspberry Pi OS
3	Gateway 3	Raspberry Pi 4 Model B 8 Gb RAM	64-BIT Quad-core Cortex-A72	8 GB LPDDR4	32 GB Kingston MicroSD	Raspberry Pi OS
4	SERVER 1	DELL OPTIPLEX 7040	Intel Core-i5 6500	32 GB DDR4	SSD M2 500 GB	UBUNTU SERVER
5	SERVER 2	CUSTOM	AMD Ryzen 5 3600	32 GB DDR4	SSD M2 1 TB	WINDOWS 10 (WSL)

Para el despliegue de todos los componentes de la arquitectura se usaron contenedores sobre la tecnología de Docker, también se usó la herramienta Docker-Compose que permite configurar fácilmente el despliegue de varios componentes, los archivos de despliegue y los archivos de configuración de cada contenedor se encuentran en el repositorio del proyecto (Jiménez & Pedraza, 2022) en la Tabla 7 se resumen las imágenes de terceros que fueron usadas para el despliegue.

Tabla 7

Contenedores de Terceros Desplegados

#	Nombre	Versión	Enlace
1	Eclipse Mosquitto	ARM64v8	Docker Hub
2	Redis	ARM64v8	Docker Hub
3	RabbitMQ	3.9.22	Docker Hub
4	MySQL	5.7	Docker Hub
5	MongoDB	4.4	Docker Hub
6	Neo4j	4.4.9	Docker Hub

Estos contenedores facilitaron el despliegue de cada escenario propuesto en el experimento, poseen configuraciones estándar para este tipo de pruebas o incluso para ser desplegados en producción.

Para el experimento propuesto se plantearon tres escenarios de prueba, el objetivo de cada escenario fue escalar progresivamente la cantidad de elementos presentes en la arquitectura y por lo tanto la cantidad de notificaciones y/o novedades que influyen en la representación de la arquitectura, para cada escenario se manejan tres momentos, la carga del escenario, los pasos de simulación y la eliminación del escenario de la arquitectura. A continuación, se presenta cada escenario mostrando la configuración de este, la visión arquitectural o de despliegue y el flujo de trabajo experimental.

8.3.1 Escenario Experimental 1

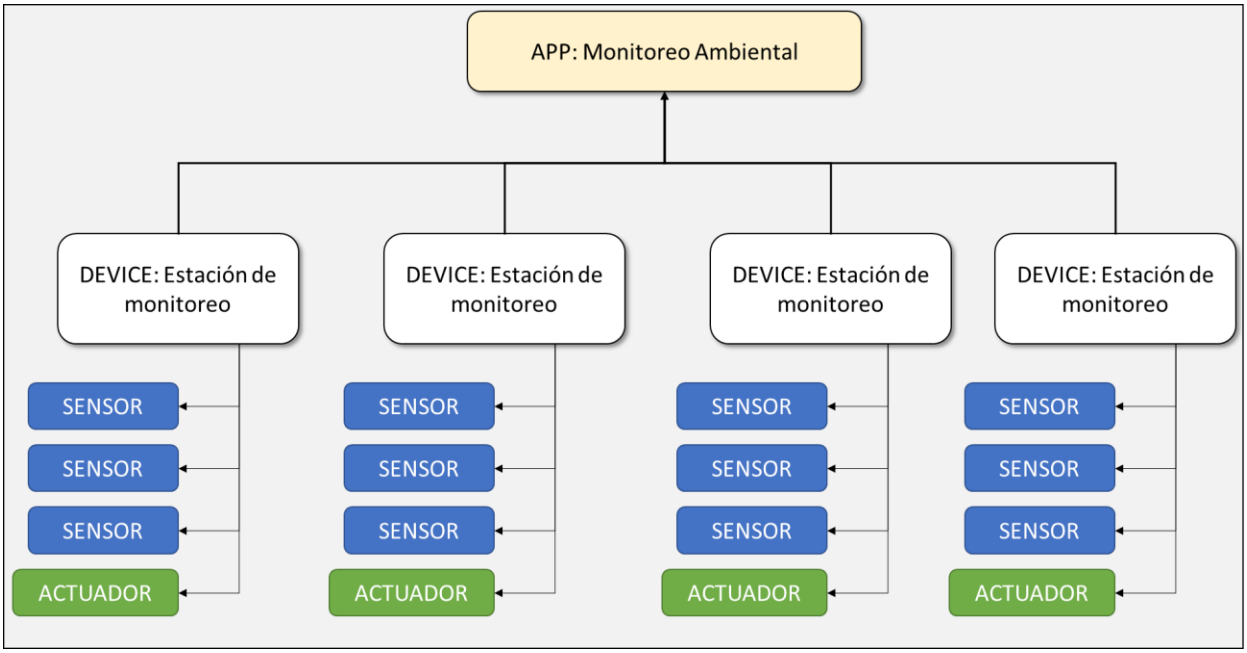
En el escenario experimental uno, el objetivo fue la creación de una aplicación de monitoreo ambiental, donde se despliegan cuatro dispositivos que son estaciones de monitoreo ambiental y que están constituidos por tres sensores y un actuador, Cada elemento tiene dos propiedades configuradas y el conocimiento de cada elemento es gestionado por el Gateway uno, que a su vez es otro dispositivo y que contiene internamente una base de datos SQL y un bróker MQTT, también se configuraron como reglas de alto nivel que la aplicación no puede tener **menos** de **tres dispositivos** y que cada dispositivo debe proporcionar al **menos cuatro recursos**.

La configuración del escenario de pruebas uno se presenta en la Figura 24, la app de monitoreo ambiental es el un elemento de la lógica de negocios del sistema desplegado en el dispositivo Gateway, aquí se configuran las cuatro estaciones de monitoreo con sus respectivos sensores y actuadores, para el escenario de pruebas las estaciones de monitoreo fueron creadas

conceptualmente en el sistema pero no se crearon las estaciones físicas ya que estaba fuera del alcance del proyecto.

Figura 24

Configuración escenario experimental 1



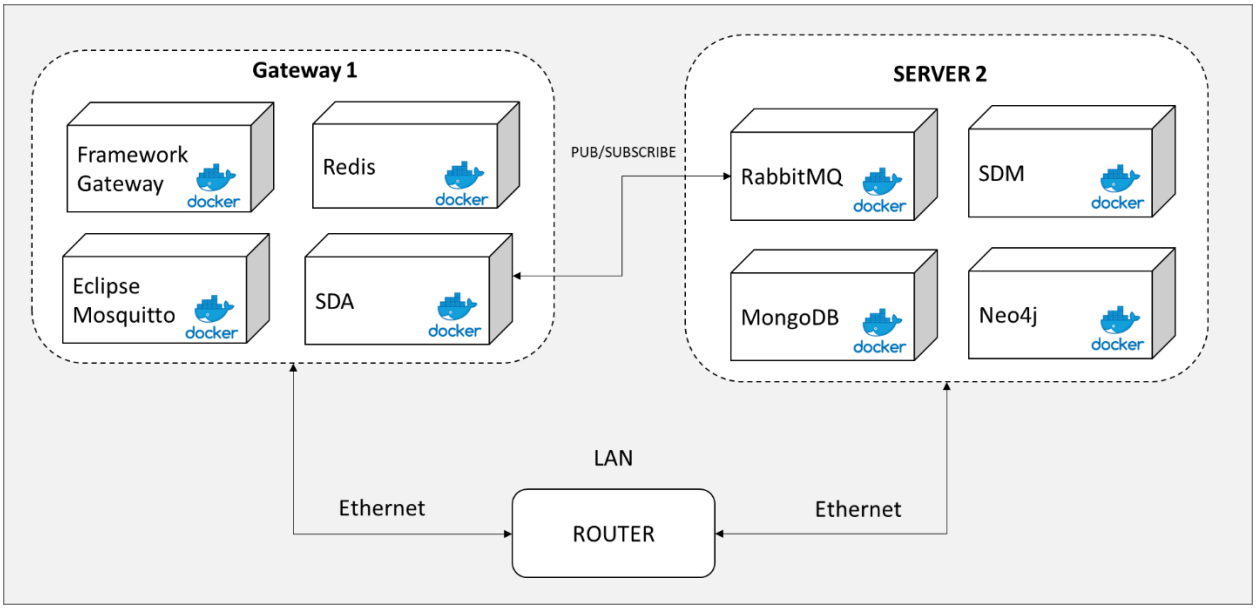
Para el despliegue del escenario experimental uno se requieren dos dispositivos hardware un Gateway y un servidor donde desplegar los mecanismos de autodescripción, en la Figura 25 se presenta la perspectiva de la arquitectura de despliegue para el escenario uno, en el Gateway 1 se desplegaron 4 contenedores, uno para el bróker interno que usa el Gateway de envío y recepción de datos con los dispositivos, el segundo contenedor corresponde al framework Gateway que se encarga de la gestión de los dispositivos, el tercer contenedor tiene desplegado Redis para el manejo de colas internas del Gateway y el cuarto contenedor contiene el mecanismo SDA.

En el server 2 también se desplegaron 4 contenedores, el primero fue el bróker RabbitMQ que permite la comunicación entre el SDA del Gateway y SDM, también fue desplegado el SDM

como un contenedor que accede a los otros dos contenedores correspondientes a la base de datos documental MongoDB y la base de datos orientada a grafos Neo4j respectivamente.

Figura 25

Vista arquitectura despliegue escenario experimental 1



Para reducir al máximo la intervención humana en el experimento y obtener resultados más uniformes se programó mediante Python un script que ejecuta el escenario en el mismo tiempo aprovechando la API-REST que expone el framework Gateway para la administración de dispositivos, el flujo de trabajo del experimento que se realizó en el escenario uno tiene tres etapas:

1) se carga el escenario de prueba, donde se crean los dispositivos, recursos, propiedades y aplicaciones en el sistema,

2) se ejecuta un conjunto de acciones definidas como eliminar, actualizar y eliminar dispositivos del sistema,

3) finalmente se elimina el escenario, borrando toda la información registrada, la evaluación de las métricas definidas se realiza en todas las etapas de la prueba.

Una vez cargado el escenario experimental uno se genera la simulación realizando los siguientes pasos:

- **Paso 1:** eliminar una estación, se debería actualizar la representación y no se generan alarmas.
- **Paso 2:** eliminar otra estación, se actualiza la representación y se debería generar una alarma por número de dispositivos.
- **Paso 3:** se crean dos estaciones nuevas, se actualiza la representación y se detiene la generación de alamas por número de dispositivos.
- **Paso 4:** se elimina el actuador de una de las estaciones, se actualiza la representación y se genera una alarma por cantidad mínima de recursos en la estación.
- **Paso 5:** se restaura el actuador en la estación, se actualiza la representación y se detiene la alarma por cantidad mínima de recursos en la estación.

Al finalizar la simulación se elimina toda la información que se creó en el sistema, cabe resaltar que la interacción debe ser únicamente con los elementos de la plataforma, el SDA y SDM deben funcionar de forma autónoma, de tal manera que cuando se crean o eliminan componentes de la arquitectura la representación muestre el estado de esta sin intervención.

El escenario experimental uno se ejecutó con 3 configuraciones, la primera vez donde todos los mecanismos funcionan adecuadamente, en la segunda ejecución el SDA se detiene durante la etapa de simulación y en la última ejecución el SDM se apaga durante la etapa de simulación, esto

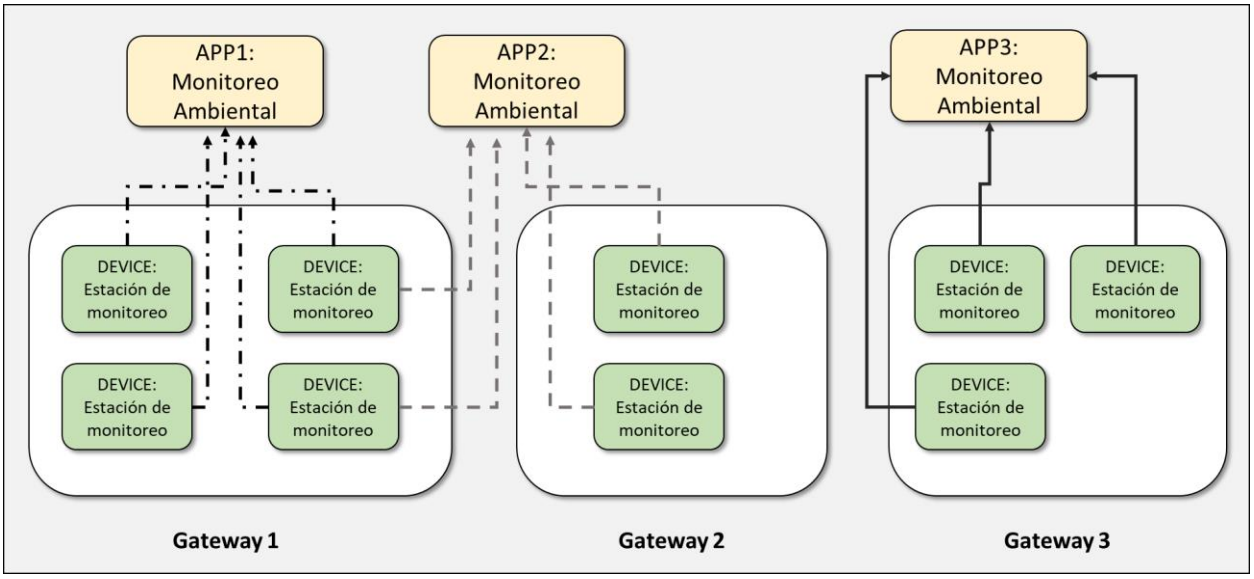
para validar la resiliencia del sistema, cada una de las configuraciones del escenario y el flujo de trabajo completo se ejecutaron tres veces para verificar que los resultados fueran similares en la misma configuración.

8.3.2 Escenario Experimental 2

En el escenario experimental dos el objetivo fue integrar varios Gateways y aplicaciones en el sistema, la configuración del escenario experimental dos se presenta la Figura 26, en cada Gateway se crearon dispositivos de estaciones de monitoreo iguales a las creadas en el escenario experimental uno (con tres sensores y un actuador) aunque el número de estaciones varió en cada Gateway, la particularidad de este escenario son las aplicaciones desplegadas, la aplicación uno tiene todos sus dispositivos en el Gateway uno y algunos son compartidos con la aplicación dos que también tiene parte de los dispositivos que usa desplegados en el Gateway dos, la aplicación tres tiene dispositivos exclusivos desplegados en el Gateway tres. Al igual que el escenario uno en el escenario experimental dos los dispositivos de las estaciones de monitoreo son conceptuales y están creados en el sistema, pero no existen físicamente, realmente para la representación del sistema es indiferente la existencia o no del dispositivo físico.

Figura 26

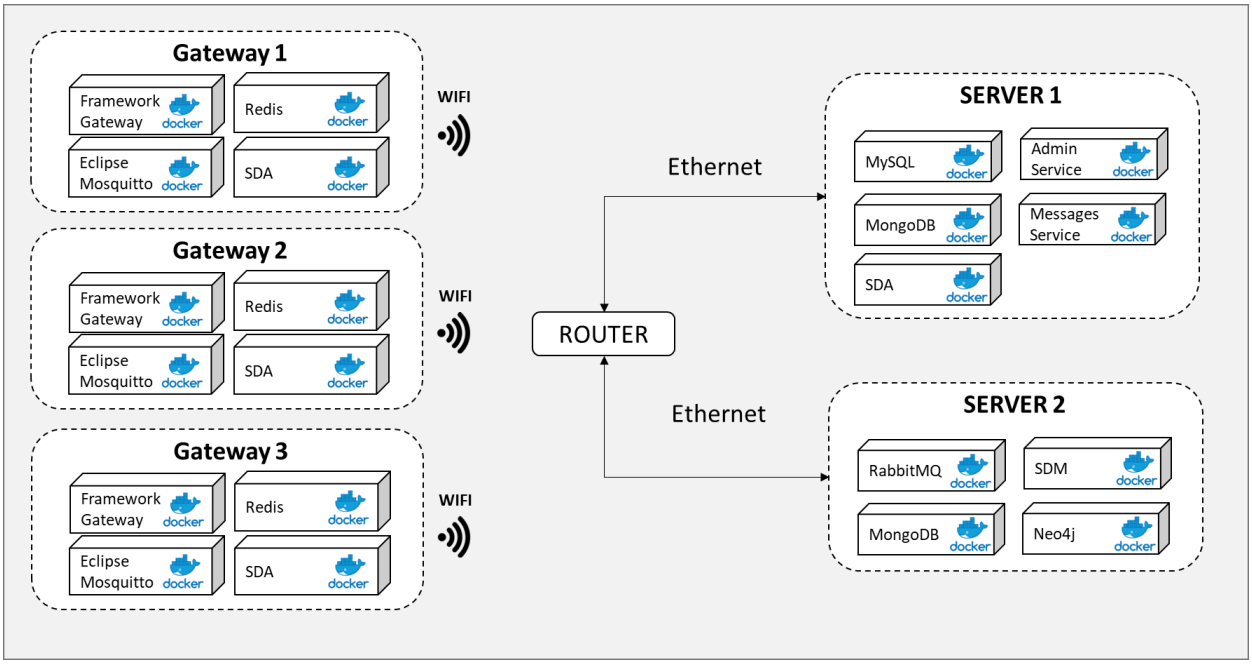
Configuración escenario experimental 2



Para el despliegue del escenario experimental dos se requieren todos los elementos hardware disponibles para la validación, En la Figura 27 se observa la visión de despliegue de la arquitectura para el escenario de pruebas dos, el despliegue en cada uno de los Gateway fue el mismo que en la topología anterior, igualmente el despliegue del SDM se realizó en el servidor dos, se adiciona el servidor uno a la topología donde se desplegaron los servicios de administración con su base de datos MySQL y el servicio de mensajes con su base de datos MongoDB, para esta topología los dispositivos Gateway funcionan por medio de una conexión WIFI, los dos servidores se encuentran conectados por cable Ethernet, el broker de mensajería RabbitMQ está desplegado en un servidor diferente al mecanismo SDA y sus bases de datos, así como los microservicios de la plataforma UIS Smart Campus, de forma que los elementos del experimento se encuentran distribuidos.

Figura 27

Vista arquitectura despliegue escenario experimental 2



Al igual que el escenario de pruebas uno, en el escenario de pruebas dos se realizó un Script de pruebas para reducir al máximo la variabilidad de los resultados, el flujo del experimento realizado en este escenario igualmente tiene tres etapas: 1) Carga del escenario, 2) Simulación, 3) Eliminación del escenario.

Para profundizar en la fase dos, una vez cargado el escenario experimental dos se genera la simulación con los siguientes pasos:

- Paso 1: Se eliminan todos los dispositivos del Gateway 2.
- Paso 2: Se elimina uno de los dispositivos del Gateway 1.
- Paso 3: Se elimina la aplicación 2.

- Paso 4: Se crean dos estaciones de monitoreo nuevas en el Gateway 2 y se crea la estación eliminada en el Gateway 1.
- Paso 5: Se crea nuevamente la aplicación 2.

Al finalizar la simulación se elimina toda la información que se creó en el sistema, cabe resaltar que en esta simulación la interacción y gestión debe ser con el servicio de administración, los Gateways se sincronizan al inicio de la prueba y luego trabajan gestionados por este servicio, el SDA y SDM deben funcionar de forma autónoma, de tal manera que cuando se crean o eliminan componentes de la arquitectura la representación muestre el estado de esta sin intervención.

El escenario experimental 2 se ejecuta con tres configuraciones, la primera vez donde todos los mecanismos funcionan adecuadamente, en la segunda ejecución el SDA del servicio de administración se detiene durante la etapa de simulación y en la última ejecución el SDM se apaga durante la etapa de simulación, esto para validar la resiliencia del sistema, en cada configuración se ejecutó tres veces el flujo de trabajo propuesto para obtener promedios y verificar la variabilidad y coherencia de los resultados.

La figura 28 muestra una foto de los dispositivos Gateway en funcionamiento durante las pruebas, estas tarjetas fueron adquiridas con la caja protectora que se observa.

Figura 28

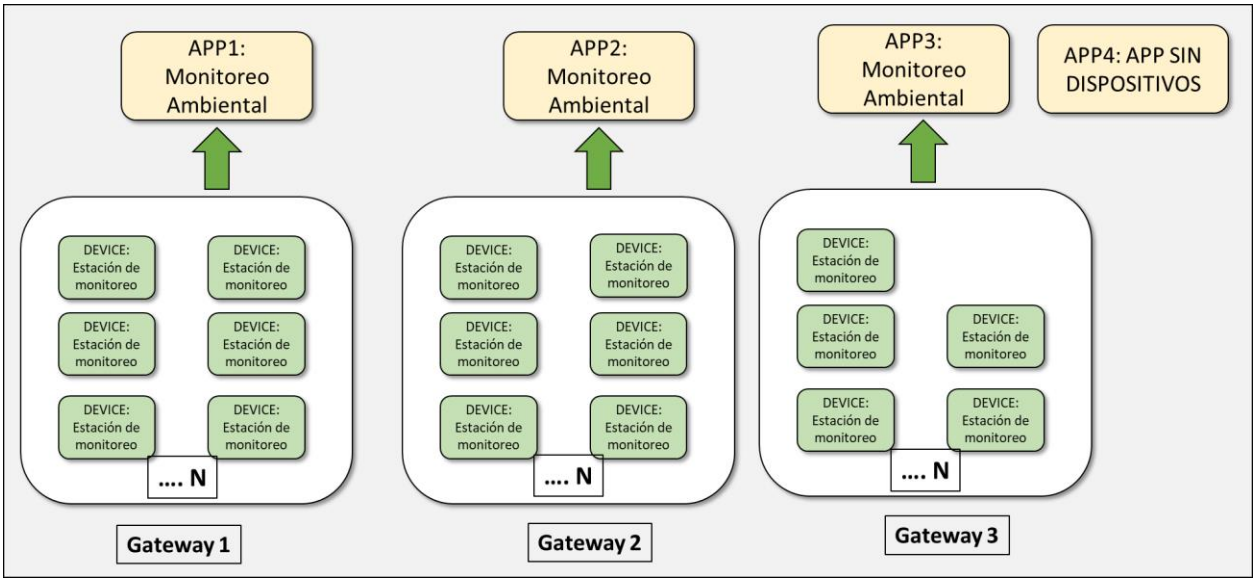
Dispositivos Gateway desplegados

8.3.3 Escenario Experimental 3

El escenario experimental tres está orientado a escalar la cantidad de componentes y dispositivos representados en la arquitectura, como se muestra en la Figura 29, se crean estaciones de monitoreo en cada Gateway, en esta configuración cada Gateway maneja dispositivos exclusivos de una aplicación y lo que se busca es aumentar la cantidad de estaciones para validar cómo evoluciona la representación del sistema cuando aumenta la escala.

Figura 29

Configuración escenario experimental 3



Este escenario usó la misma arquitectura de despliegue del escenario experimental dos que puede ser apreciada en la Figura 27, al igual que en los dos escenarios anteriores se realizó un script para generar las solicitudes y reducir la variabilidad entre los tiempos de las solicitudes, los tiempos de solicitud y respuesta fueron medidos nuevamente por el mismo script en el mismo equipo para garantizar el reloj del sistema, el flujo de trabajo propuesto para este escenario solo incluye dos fases, la carga y descarga del escenario de pruebas, variando la cantidad de dispositivos o estaciones desplegadas en cada Gateway.

8.4 Análisis de resultados

En este apartado se presenta el análisis de resultados realizado para cada escenario de pruebas propuesto.

8.4.1 Resultados escenario experimental 1

El objetivo de las pruebas realizadas en el escenario experimental 1 fue validar la coherencia de la representación al usar solo el dispositivo Gateway 1 con el despliegue de los contenedores y el SDA y el servidor que contiene el SDM con sus respectivas bases de datos y el bróker RabbitMQ, en la Tabla 8 se observan los resultados obtenidos, cada prueba fue repetida 3 veces para verificar el resultado, las pruebas consistieron en cargar el escenario propuesto, realizar la simulación y posteriormente borrar el escenario, el número de elementos totales es 21.

Tabla 8

Resultados escenario experimental 1

#	Retraso en la sincronización (segundos)	Tasa de Pérdida (Novedades Perdidas/Totales)	Tasa de Procesamiento (Novedades x Segundo)	Tasa de Coherencia (Elementos bien representados/Total)	Componentes no recuperados ante una falla.	Duración de la prueba Segundos
1	1,53	0/258 (0%)	5,16	21/21 (100%)	N/A	50
2	1,96	0/258 (0%)	4,87	21/21 (100%)	N/A	53
3	2,62	0/258 (0%)	5,16	21/21 (100%)	N/A	50
PROM	**2,04**	**0/258 (0%)**	**5,06**	**21/21 (100%)**	**N/A**	**51**
4	4,67	0/258 (0%)	2,84	21/21 (100%)	0	91
5	2,80	0/258 (0%)	3,74	21/21 (100%)	0	69
6	2,46	0/258 (0%)	3,23	21/21 (100%)	0	80
PROM	**3,31**	**0/258 (0%)**	**3,27**	**21/21 (100%)**	**0**	**80**
7	3,29	0/258 (0%)	4,10	21/21 (100%)	0	63
8	3,54	0/258 (0%)	3,97	21/21 (100%)	0	65
9	4,19	0/258 (0%)	3,11	21/21 (100%)	0	83
PROM	**3,67**	**0/258 (0%)**	**3,72**	**21/21 (100%)**	**0**	**70**

Los primeros tres registros de la Tabla 8 corresponden a la prueba donde no fallaba ningún componente, se puede observar que el tiempo promedio de respuesta fue de 2,04 segundos, no se

perdieron novedades reportadas, se procesaron en promedio 5 novedades por segundo, la coherencia de los elementos representados es del 100% y en total esta prueba tuvo una duración de 51 segundos.

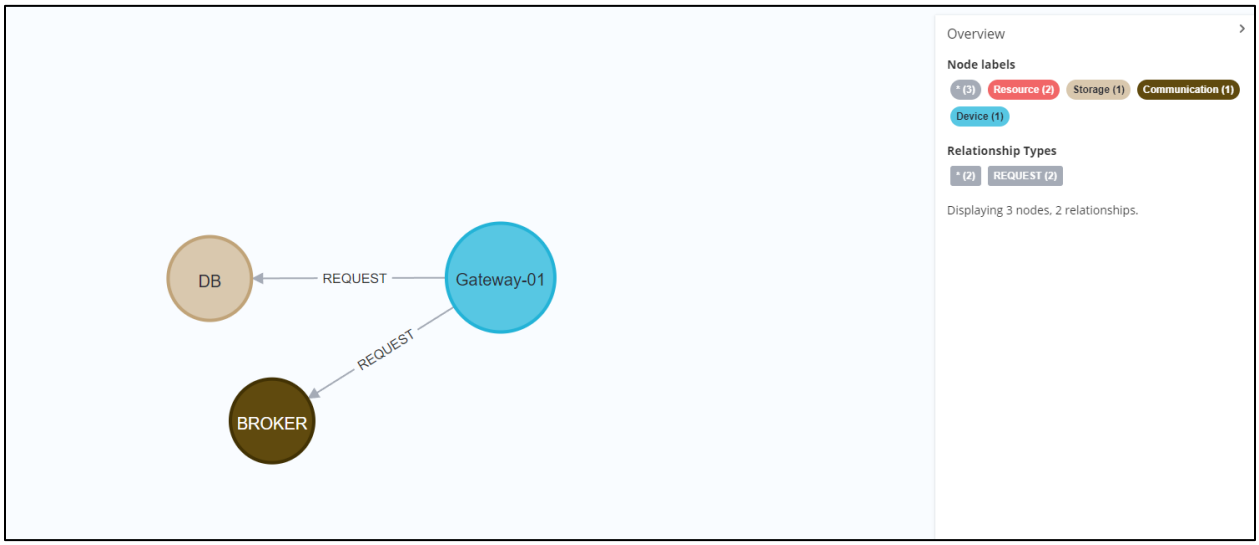
Los registros 4, 5 y 6 pertenecen a la prueba donde se desactivo el SDA durante la fase de simulación, en esta prueba el tiempo promedio de respuesta fue de 3.31 segundos, aumento como cabía de esperarse ya que al no estar disponible el SDA durante la fase de simulación (aproximadamente 30 segundos), todas las novedades que se generaron en ese lapso fueron almacenadas en cola a la espera de ser procesadas, esto también ocasiono una disminución en la cantidad de solicitudes evaluadas por segundo, pero gracias a las colas de mensajes asíncronas aunque el SDA no esté disponible durante un tiempo, el sistema de representación se recupera cuando está de nuevo en línea y no se presentan incoherencias ya que se conserva el orden de generación de las novedades. En los registros 7, 8 y 9 se observan los registros del escenario 1, donde el SDM falla durante la fase de simulación (aproximadamente 30 segundos), al igual que en los registros donde falla el SDA, se mantienen las novedades en cola y cuando se recupera el SDM son procesadas, esto aumenta el tiempo de respuesta para algunas de las solicitudes y disminuye el promedio de solicitudes procesadas por segundo.

Durante la ejecución del escenario de pruebas en las 9 mediciones que se realizaron el promedio de uso de CPU del Gateway 1 fue de 36.21% y el consumo de memoria RAM fue de 17.45%, lo cual demuestra que los mecanismos desplegados junto con el framework Gateway no generan una carga exhaustiva que consuma todos los recursos del dispositivo Gateway 1, al menos en un escenario con pocos dispositivos como cabe esperarse.

La coherencia del escenario fue medida en cada una de las fases propuestas de la simulación, el grafo de conocimiento generado se presenta en la siguiente figura:

Figura 30

Grafo de conocimiento antes de cargar el escenario 1



En la Figura 30 se observa el grafo de conocimiento que representa el Gateway sin ningún dispositivo asociado, tiene dos recursos, el recurso de almacenamiento que es la base de datos interna, y el recurso de comunicación que es el bróker de Eclipse Mosquitto registrado, para el análisis del escenario durante la consulta se omitieron estos nodos ya que no varían en ninguna fase de la simulación.

En la Figura 31 y Figura 32 se observa el grafo de conocimiento generado una vez cargado el escenario 1 sobre el dispositivo Gateway 1, aquí se puede observar la existencia de 4 dispositivos los cuales tienen nombres generados secuencialmente por el código de la prueba, igualmente se aplica para cada uno de los recursos asociados, esta representación fue obtenida mediante la interfaz web que expone Neo4j.

Figura 31

Resumen de nodos y relaciones escenario 1

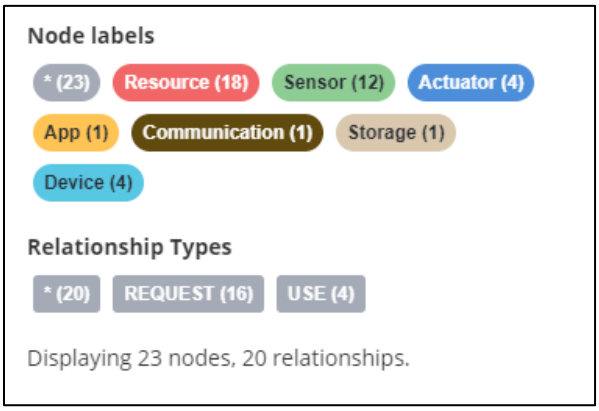
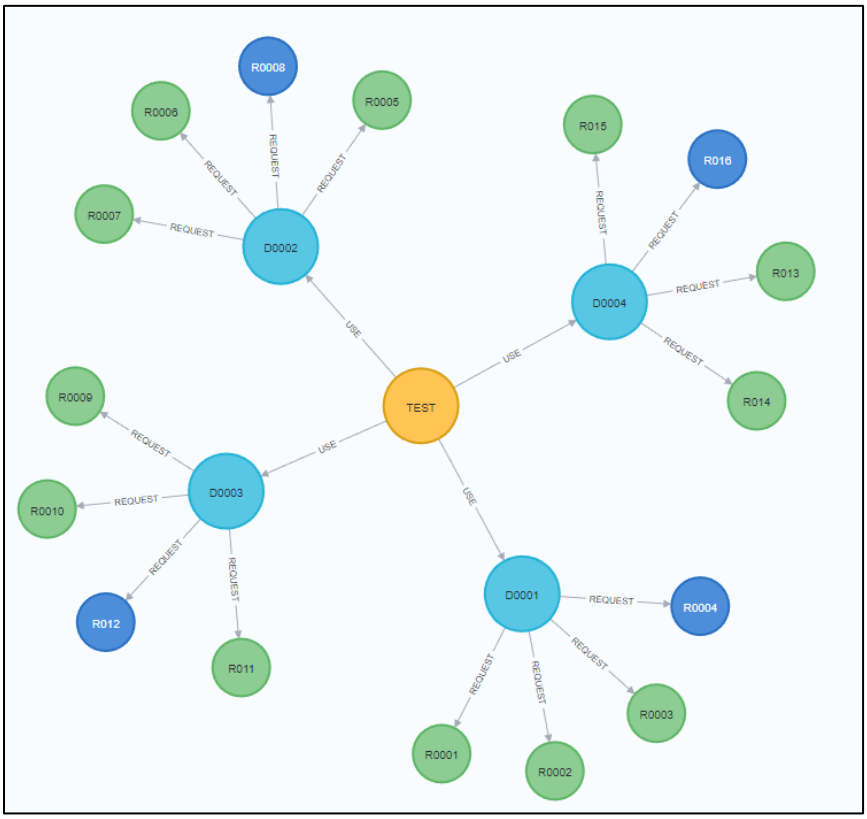


Figura 32

Grafo de conocimiento escenario experimental 1



Fue posible validar la sincronización y coherencia de la representación dada por el grafo de conocimiento ya que en cada paso de la simulación el grafo describe correctamente el estado actual del sistema, las alarmas establecidas y generadas por el sistema conforme a las políticas de alto nivel se guardaron correctamente en los logs del sistema y al consultarse se ven como en la Figura 33, aquí se observa el id del dispositivo o aplicación que no cumple con las políticas establecidas y la fecha en la cual se generó la advertencia.

Figura 33

Log de Advertencia de Sistema

```
The APP with GID: 1 doesn't have the required devices.| 2022-09-12 23:06:50
The device with GID: 2 doesn't have the required resources. | 2022-09-12 23:07:10
```

Se puede concluir de los resultados obtenidos en la ejecución del escenario experimental 1 que los mecanismos implementados y desplegados en el Gateway no generan un estrés que pueda afectar el normal funcionamiento del dispositivo, adicionalmente la representación obtenida estuvo acorde al escenario simulado dentro de la arquitectura, con tiempos de respuesta óptimos y tolerantes a fallas, debido a la escala de los elementos que se presenta en este escenario no se encontraron problemas de coherencia en la representación generada.

8.4.2 Resultados escenario experimental 2

El objetivo de las pruebas realizadas en el escenario experimental dos fue validar la integración de todos los elementos presentes en la arquitectura evaluada y la correcta sincronización de la representación generada con el estado de un sistema distribuido. En la Tabla 9 se puede observar los resultados de la evaluación en las diferentes métricas establecidas, se resalta que la coherencia del sistema se mantiene en el 100% incluso en los escenarios donde falla

algún componente, los tiempos de sincronización de la representación siguen siendo buenos manteniéndose cerca al segundo, lo cual quiere decir que a los mecanismos de autodescripción les toma 1 segundo desde que se genera un cambio en el sistema hasta que se actualiza la representación, igualmente al escenario experimental 1, los registros 1,2 y 3 corresponden a las medidas sobre el sistema cuando ningún componente falla, los registros 4,5 y 6 corresponden a la falla del SDA durante la fase de simulación, finalmente los registros 7, 8 y 9 corresponden a la falla del SDM durante la fase de simulación.

Tabla 9

Resultados escenario experimental 2

#	Retraso en la sincronización (segundos)	Tasa de Pérdida (Novedades Perdidas/Totales)	Tasa de Procesamiento (Novedades x Segundo)	Tasa de Coherencia (Elementos bien representados/Total)	Duración de la prueba Segundos	Componentes no recuperados ante una falla.
1	1,20	0/564 (0%)	5,81	61/61 (100%)	97	N/A
2	1,18	0/564 (0%)	5,76	61/61 (100%)	98	N/A
3	1,31	0/564 (0%)	5,64	61/61 (100%)	100	N/A
PROM	**1,23**	**0/564 (0%)**	**5,74**	**61/61 (100%)**	**98**	**N/A**
4	1,2	0/564 (0%)	4,44	61/61 (100%)	127	0
5	1,19	0/564 (0%)	4,95	61/61 (100%)	114	0
6	1,22	0/564 (0%)	4,70	61/61 (100%)	120	0
PROM	**1,20**	**0/564 (0%)**	**4,69**	**61/61 (100%)**	**120**	**0**
7	1,16	0/564 (0%)	4,66	61/61 (100%)	121	0
8	1,20	0/564 (0%)	4,86	61/61 (100%)	116	0
9	1,18	0/564 (0%)	4,82	61/61 (100%)	117	0
PROM	**1,18**	**0/564 (0%)**	**4,78**	**61/61 (100%)**	**118**	**0**

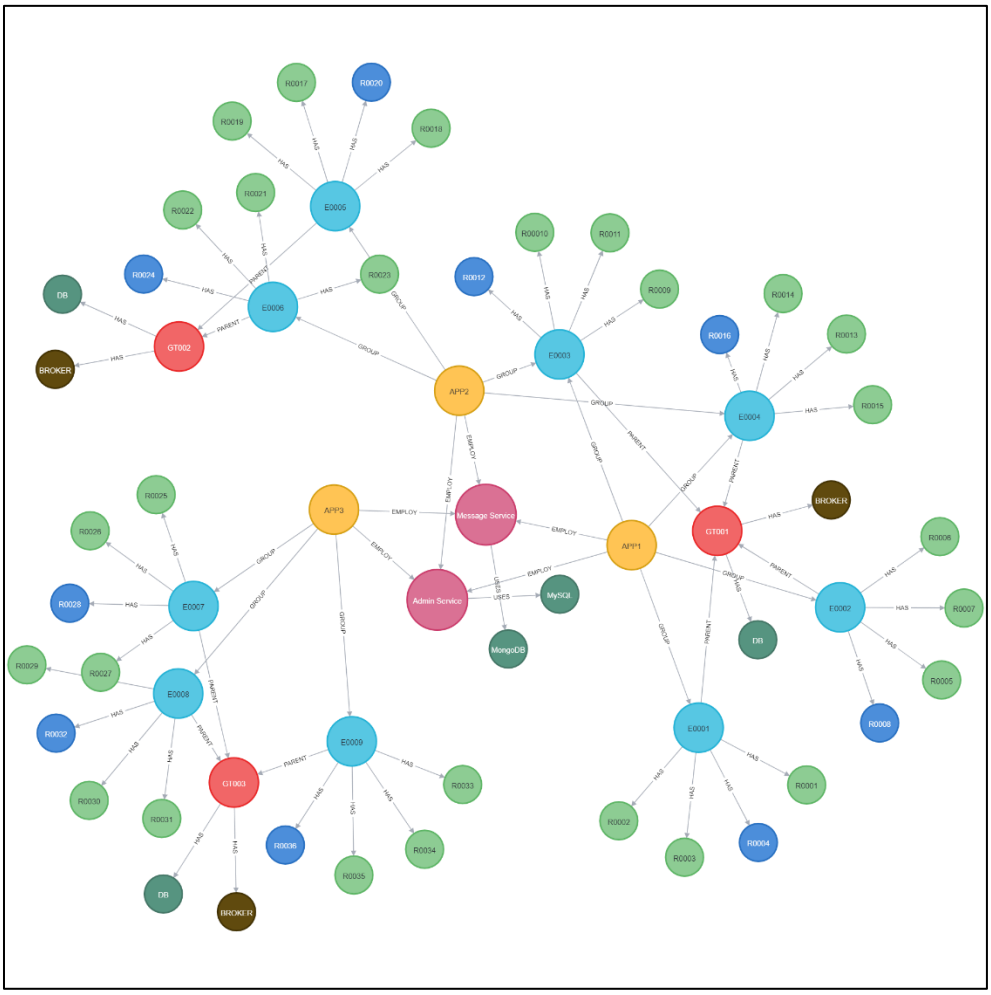
En esta prueba también se registraron el uso promedio de CPU y memoria RAM en los dispositivos Gateway, para el Gateway 1 que tiene solo 4 Gb de memoria RAM el uso promedio

de CPU se situó en 4.9 %, respecto al escenario 1 el uso de CPU del Gateway 1 disminuyo considerablemente ya que no era el único dispositivo encargado de procesar todas las solicitudes y al estar sincronizado con los servicios de la plataforma la carga de trabajo disminuye, el uso de memoria RAM también disminuyo respecto al escenario anterior registrando un uso promedio de 10.2 %, para los Gateway 2 y 3 que tienen mayor capacidad de memoria RAM, el uso promedio de CPU fue de 3% y el consumo de memoria RAM se sitúa en el 6.6% lo cual es coherente con los resultados del Gateway 1.

El grafo de conocimiento obtenido, sus nodos y relaciones son presentados en la Figura 34, ahí podemos apreciar la representación de los servicios (nodos rosas), las aplicaciones (nodos naranjas), los dispositivos (nodos celestes), los gateways (nodos rojos), los sensores (nodos verdes claro), los actuadores (nodos azules), brokers (nodos marrones) y bases de datos (nodos verdes oscuro).

Figura 34

Grafo de conocimiento escenario de experimental 2



De los resultados obtenidos en este escenario se resalta la correcta generación del grafo de conocimiento y/o representación de la arquitectura, incluso cuando está asociado a una arquitectura distribuida, la capacidad de los mecanismos de autodescripción para procesar las novedades incluso si alguno de estos es dado de baja durante un tiempo y el rendimiento es satisfactorio.

8.4.3 Resultados escenario experimental 3

En el escenario experimental tres el objetivo fue evaluar la escalabilidad del sistema y las limitantes de la representación obtenida, en este escenario se evaluó la creación de 100 componentes en la arquitectura y posteriormente se realizó la evaluación para 500 componentes principalmente recursos y dispositivos, en la Tabla 10 se presentan los resultados del escenario experimental tres, se resalta la coherencia de la representación que se mantiene incluso con 500 componentes, el retraso en la sincronización de la representación sigue por el orden de 1 segundo lo cual es un resultado bastante bueno.

Tabla 10

Resultados escenario experimental 3

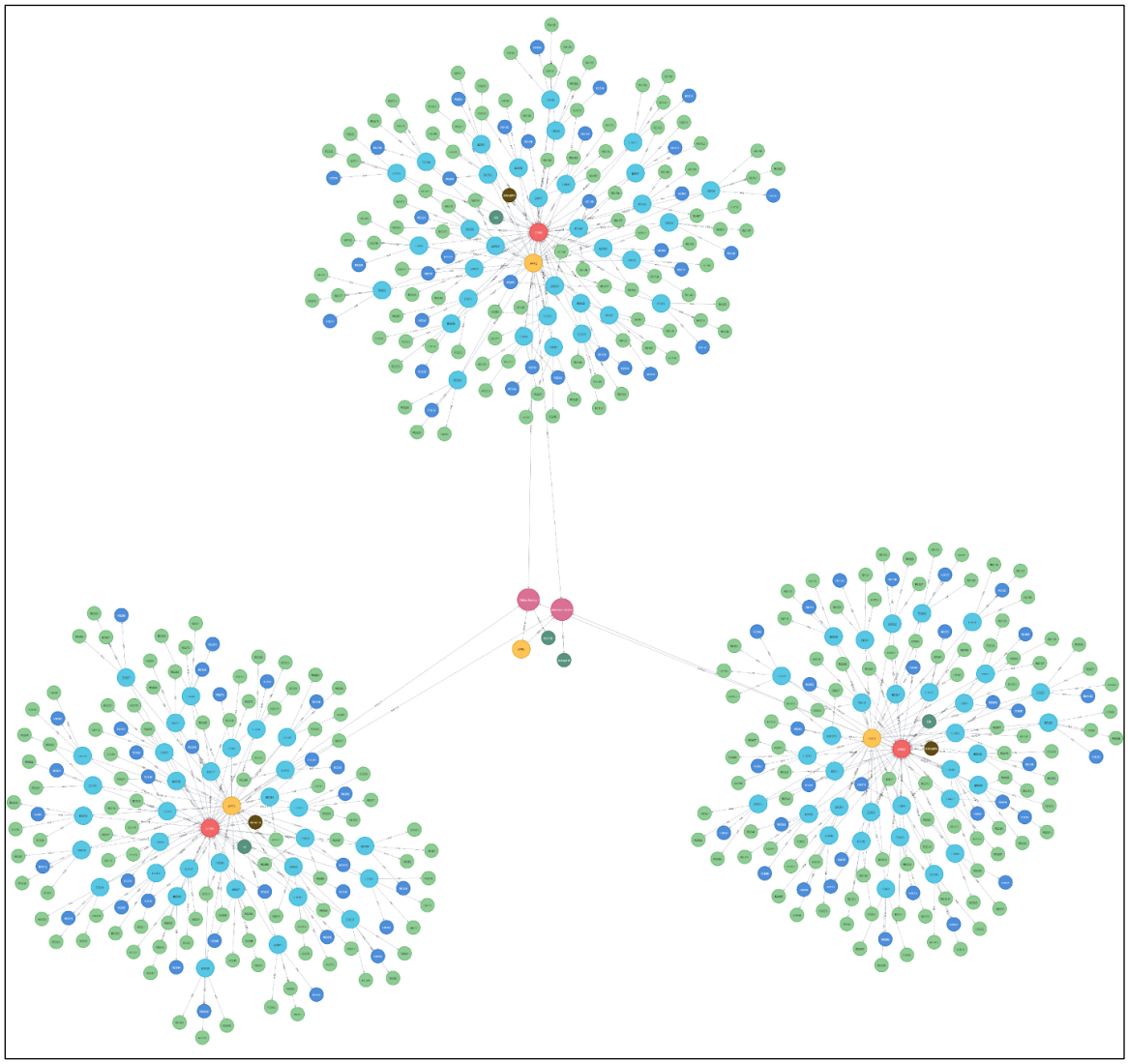
#	**1 -** Retraso **en la sincronización (segundos)**	**2 - Tasa de Pérdida (Novedades Perdidas/Totales)**	**3.1 Tasa de Procesamiento (Novedades x Segundo)**	**3.2 Tasa de** Coherencia **(Elementos bien representados/Total)**	**Duración de la prueba Segundos**
1	1,20	0/751 (0%)	5,53	100/100 (100%)	102
2	1,26	0/751 (0%)	5,42	100/100 (100%)	104
PROM	**1,23**	**0/751 (0%)**	**5,48**	**100/100 (100%)**	**103**
1	1,18	0/4110 (0%)	7,12	500/500 (100%)	577
2	1,16	0/4110 (0%)	7,27	500/500 (100%)	565
PROM	**1,17**	**0/4110 (0%)**	**0,99**	**500/500 (100%)**	**571**

El grafo de conocimiento generado para el escenario 3 y la evaluación de 500 componentes representados se observa en la Figura 35, aquí es evidente cómo se organiza el grafo en tres estructuras conectadas únicamente por los servicios, cada Gateway maneja aproximadamente 30 dispositivos cada uno es una estación de monitoreo (con 3 sensores y 1 actuador). De este escenario

se puede resaltar que la representación visual del grafo está adaptada para mostrar máximo 300 elementos, una vez se supera este umbral se tiene que modificar la configuración de Neo4j para que permita visualizar más nodos, la visualización generada es bastante lenta después de superar el umbral, se recomienda mejor realizar consultas especializadas en zonas específicas para una mejor visualización de la arquitectura.

Figura 35

Grafo de conocimiento escenario 3 - 500 componentes



Con las pruebas realizadas fue posible validar la viabilidad del diseño propuesto, los prototipos implementados presentaron resultados conforme a los objetivos propuestos y representan una herramienta que puede apoyar en la administración de arquitecturas IoT distribuidas, las pruebas realizadas no presentan fallas de coherencia incluso en el escenario más grande, esto no quiere decir que el prototipo propuesto no pueda **presentar fallos** lo cierto es que al tener solo en cuenta la correcta representación de los componentes se está omitiendo la validación de la correcta representación de las relaciones, estas relaciones en su mayoría se crearon y representaron bien, pero **se observaron fallas** a la hora de eliminar el escenario, ya que para el escenario experimental 3 cada vez que se eliminaba el escenario quedaban algunas relaciones mapeadas en MongoDB incluso si el elemento ya no pertenecía a la plataforma. También es posible que en escenarios más estresantes y extensos se encuentren nuevas fallas del prototipo por lo tanto se recomienda ampliar la experimentación para obtener componentes que puedan ser llevados a producción.

9. Conclusiones y Trabajo Futuro

El trabajo de investigación desarrollado propone un enfoque diferente para la representación de arquitecturas IoT distribuidas, lo suficientemente robusto y escalable para ser desplegado en arquitecturas con cientos o incluso miles de dispositivos.

El modelo conceptual propuesto cumple con las necesidades del problema y aunque pudiera ser ampliado para incluir nociones presentes en otros contextos, contiene los conceptos y relaciones necesarias para representar la mayoría de las arquitecturas IoT generales del mercado.

Los mecanismos de autodescripción implementados cumplieron su objetivo y permitieron generar representaciones coherentes del sistema haciendo uso del modelo conceptual, el flujo de novedades propuesto es robusto y permite al sistema trabajar bajo estrés incluso si un mecanismo sufre una falla temporal.

Las tecnologías usadas en la implementación son lo bastante robustas para funcionar en un entorno distribuido con la suficiente rapidez y un uso considerable de recursos, Python como lenguaje de programación permitió prototipar rápidamente las soluciones propuestas.

Las representaciones obtenidas son los suficientemente robustas, rápidas y escalables de forma que puedan ser herramientas de utilidad para la administración de arquitecturas IoT distribuidas, adicionalmente la mayor parte de la generación de dicha representación se genera de forma autónoma, por lo cual se reduce la intervención humana en la administración del sistema y se concentra en el estudio y definición de políticas de alto nivel.

Los grafos de conocimiento son representaciones muy útiles al ser implementadas en tecnologías como las bases de datos orientadas a grafos, es importante tener en cuenta que su

utilidad más allá de generar una representación visual es la de realizar el análisis de información sobre miles o millones de nodos representados.

Es viable explorar otros enfoques a futuro que permitan dotar de más capacidades a los mecanismos definidos, por ejemplo, la definición de ontologías permitiría poder implementar inferencias más complejas en el modelo de conocimiento del sistema, incluso pudiendo generar nuevo conocimiento a partir del conocimiento existente.

Es importante adaptar el modelo propuesto con algún estándar de representación ampliamente usado, de forma que las nociones expresadas sean compatibles con las arquitecturas IoT que sigan dicho estándar y que siga siendo vigente en el tiempo.

Se recomienda realizar pruebas más extensas a futuro si se requiere llevar los mecanismos a un sistema en producción, ya que los experimentos realizados en esta investigación solo validaron la correcta generación de elementos, pero no fueron extensos en el tiempo.

Referencias Bibliográficas

Aazam, M., Zeadally, S., & Harras, K. (2018). Fog computing architecture, evaluation, and future research directions. *IEEE Communications Magazine*, 46-52.

Abdellaoui, G., Megnafi, H., & Bendimerad, F. (2020). A novel model using Reo for IoT self-configuration systems. *2020 1st International Conference on Communications, Control Systems and Signal Processing*, 1-5.

Abdmeziem, M., Tandjaoui, D., & Romdhani, I. (2016). Architecting the Internet of Things: State of the Art. *Robots and Sensor Clouds. Studies in Systems, Decision and Control., 36*, 55-75.

AI-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, 2347-2376. doi:10.1109/COMST.2015.2444095.

Alberti, A. M., & Singh, D. (2013). Internet of things: perspectives, challenges and opportunities. *International Workshop on Telecommunications (IWT 2013)*, 1-6.

Alfonso, I., Garcés, K., Castro, H., & Cabot, J. (2021). Modeling self-adaptative IoT architectures. *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 761-766.

Al-Qaseemi, S., Almulhim, H., Almulhim, M., & Chaudhry, S. (2016). IoT architecture challenges and issues: Lack of standardization. *2016 Future Technologies Conference (FTC)*, 731-738. doi: 10.1109/FTC.2016.7821686

Alulema, D., Criado, J., & Iribarne, L. (2019). A Model-driven Approach for the Integration of Hardware Nodes in the IoT. *In World Conference on Information Systems and Technologies*, 801-11.

Alvarez, G. (2019). *Control automático adaptativo por medio del ciclo MAPE-K, en el Internet de las cosas, usando el System On Chip ESP32.* Manizales: Universidad Autónoma de Manizales.

Amazon Web Services. (2020). *AWS Industrial IoT Predictive Maintenance ML Model Reference Architecture*. Obtenido de https://d1.awsstatic.com/architecture-diagrams/ArchitectureDiagrams/aws-industrial-PdM-ML-storage-RA.pdf

Amazon Web Services. (2022). *AWS IoT*. Recuperado el 15 de 08 de 2022, de https://aws.amazon.com/es/iot/

Ariza, J., Garcés, K., Cardozo, N., Rodríguez, J., & Jiménez, F. (2021). IoT architecture for adaptation to transient devices. Journal of Parallel and Distributed Computing. *Journal of Parallel and Distributed Computing*, 14-30.

Ariza, J., Mendoza, C., Garcés, K., & Cardozo, N. (2018). A research agenda for iot adaptive architectures. *Multidisciplinary Digital Publishing Institute Proceedings*, 1229.

Ashton, K. (2009). That ‘internet of things’ thing. *RFID journal*, 97-114.

Banks, C., & Sokolowski, J. (2010). *Modeling and simulation fundamentals: theoretical underpinnings and practical domains.* Wiley.

Beneteau, E., Richards, O., Zhang, M., Kientz, J., Yip, J., & Hiniker, A. (2019). Communication Breakdowns Between Families and Alexa. *In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*, 1–13. doi:https://doi.org/10.1145/3290605.3300473

Bilal, K., Khalid, O., Edbad, A., & Khan, S. (2018). Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Computer Networks, 130*, 94-120. doi:https://doi.org/10.1016/j.comnet.2017.10.002

Cervera, J. P. (2021). Conectividad de Internet en Colombia y su relación con los Objetivos de Desarrollo Sostenible (2015-2020). *Ciencia Y Poder Aéreo, 16*, 39–5. doi:https://doi.org/10.18667/cienciaypoderaereo.705

Chen, X., Jia, S., & Xiang, Y. (2020). A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications*.

Cheruvu, S., Kumar, A., Smith, N., & Wheeler, D. (2020). IoT frameworks and complexity. *Demystifying Internet of Things Security, 23-148.*, 32-35.

CISCO. (2014). *The Internet of Things Reference Model*. Obtenido de https://www.cisco.com/c/dam/global/en_ph/assets/ciscoconnect/pdf/bigdata/jim_green_cisco_connect.pdf

Čolaković, A., & Hadžialić, M. (2018). Internet of things (iot): A review of enabling technologies, challenges, and open research issues. *Computer Networks, 144*, 17-39.

Costa, B., Pires, P., & Delicato, F. (2019). Modeling SOA-based IoT applications with SoaML4IoT. *IEEE 5th World Forum on Internet of Things*, 496-501.

Dadkhah, M., Lagzian, M., Rahimnia, F., & Kimiafar, K. (2020). What do websites say about internet of things challenges? A text mining approach. *Science & Technology Libraries, 39*(2), 125-141. doi:https://doi.org/10.1080/0194262X.2020.1715320

Departamento Administrativo Nacional de Estadística – DANE. (2021). *Indicadores básicos de tenencia y uso de Tecnologías de la Información y las Comunicaciones – TIC en hogares y personas de 5 y más años de edad Departamental.* Obtenido de www.dane.gov.co

Erazo, L., Cedillo, P., Rossi, G., & Moyano, J. (2022). A Domain-Specific Language for Modeling IoT System Architectures That Support Monitoring. *IEEE Access, 10*, 61639-61665. doi:10.1109/ACCESS.2022.3181166

Evans, J. (2020). Social computing unhinged. *Journal of Social Computing*, 1-13.

Gabrielson, J. (2019). *Los desafíos de los sistemas distribuidos*. Obtenido de Amazon Builde's Library: https://aws.amazon.com/es/builders-library/challenges-with-distributed-systems/?did=ba_card&trk=ba_card

González, C., Meana, D., García, V., Jiménez, A., & Petearson, J. (2020). Midgar: Creation of a graphic domain-specific language to generate smart objects for internet of things scenarios using model-driven engineering. *IEEE Access*, 141872-141894.

Google. (2022). *IoT Core*. Recuperado el 15 de 08 de 2022, de https://cloud.google.com/iot-core

Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., & Reinfurt, L. (2016). Comparison of IoT platform architectures: A field study based on a reference architecture. *Cloudification of the Internet of Things (CIoT)*, 1-6. doi: 10.1109/CIOT.2016.7872918

Hajam, S. S., & Sofi, S. A. (2021). IoT-Fog architectures in smart city applications: A survey. *China Communications*, 117-140.

Haller, S., Serbanati, A., Bauer, M., & Carrez, F. (2013). A domain model for the internet of things. *IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, 411-417.

Horn, P. J. (2001). Autonomic Computing: IBM's Perspective on the State of Information Technology.

IBM. (2006). An architectural blueprint for autonomic computing. *IBM White Paper*.

International Telecommunication Union. (2005). The internet of things. *ITU Internet report*.

ISO. (Agosto de 2018). *Internet of Things (IoT) - Reference Architecture, Standard.* Recuperado el 15 de 07 de 2022, de www.iso.org: https://www.iso.org/standard/65695.html

Jacob, B., Lanyon-Hogg, R., Nadgir, D., & Yassin, A. (2004). *A Practical Guide to the IBM Autonomic Computing Toolkit RedBook.* Tech Rep.

Jiménez H., P. G. (20 de 09 de 2022). Repositorios Smart Campus UIS. Bucaramanga, Colombia. Obtenido de https://github.com/UIS-IoT-Smart-Campus

Jiménez, H., & Pedraza, G. (05 de 09 de 2022). Repositorio Público Smart Campus UIS. *GitHub*. Bucaramanga. Obtenido de https://github.com/UIS-IoT-Smart-Campus

Jiménez, H., Cárcamo, E., & Pedraza, G. (2020). Plataforma Software Extensible para Campus Inteligente Basada en Microservicios. *Revista Ibérica de Sistemas e Tecnologias de Informação.*, 270-282.

Lalanda, P., McCann, J., & Diaconescu, A. (2013). *Autonomic computing, principles design and implementation.* Springer.

Le-Dang, Q., & Le-Ngoc, T. (2019). Scalable Blockchain-based Architecture for Massive IoT Reconfiguration. *IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, 1-4. doi:10.1109/CCECE.2019.8861858.

Lewis, P., Platzner, M., Rinner, B., Torresen, J., & Yao, X. (2016). *Self-Aware Computing Systems. Natural Computing Series.* Springer.

Liu, Y., Fieldsend, J., & Min, G. (2017). A Framework of Fog Computing: Architecture, Challenges, and Optimization. *IEEE Access*, 25445-25454. doi:10.1109/ACCESS.2017.2766923

Lombardi, M., Pascale, F., & Santaniello, D. (2021). Internet of things: A general overview between architectures, protocols and applications. *Information*, 87.

Lopez, F. (2015). *Sistemas distribuidos.*

Mazon, B., & Pan , A. (2021). Internet of Things: State-of-the-art, Computing Paradigms and Reference Architectures. *IEEE Latin America Transactions*, 49-63.

Microsoft. (2022). *Arquitectura de referencia de Azure IoT*. Recuperado el 20 de 07 de 2022, de learn.microsoft: https://learn.microsoft.com/es-es/azure/architecture/reference-architectures/iot

Misquitta, L. (2022). *Organization Learning*. Recuperado el 15 de 08 de 2022, de Graph Gist: https://portal.graphgist.org/graph_gists/organization-learning

Naciones Unidas. (2015). Transformar nuestro mundo: La Agenda 2030 para el desarrollo Sostenible. *Resolución probada por la Asamblea General*, 3-40.

Nami, M., & Bertels, K. (2007). A survey of autonomic computing systems. *Thrid International Conference on Autonomic and Autonomous Systems*, 26-26.

Negash, B., Westerlund, T., Rahmani, A., Lijieber, P., & Tenhunen, H. (2017). DoS-IL: A domain specific Internet of Things language for resource constrained devices. *Procedia computer science.*, 416-423.

Neo4j Inc. (2021). *Neo4j Documentation*. Recuperado el 28 de 08 de 2022, de https://neo4j.com/docs/

Ngu, A., Gutierrez, M., Metsis, V., Nepal, S., & Sheng, Q. (2016). IoT Middleware: A Survey on Issues and Enabling Technologies. *IEEE Internet of Things Journal,, 4*(1), 1-20. doi:10.1109/JIOT.2016.2615180

Pico, P., Holgado, J., Sánchez, D., & Sampietro, J. (2018). Towards the internet of agents: an analysis of the internet of things from the intelligence and autonomy perspective. *Universidad Nacional de Colombia - Sede Bogotá - Facultad de Ingeniería.*

Pierleoni, P., Concetti, R., Belli, A., & Palma, L. (2019). Amazon, Google and Microsoft solutions for IoT: Architectures and a performance comparison. *IEEE access*, 5455-5470.

Poslad, S. (2011). *Ubiquitous computing: smart devices, environments and interactions.* John Wiley & Sons.

Sampaio, H., Koch, F., Becker, C., Boing, R., & Santa Cruz, R. (2019). Autonomic management of power consumption with iot and fog computing. *International Conference on Green, Pervasive, and Cloud Computing*, 89-103.

Senyo, P., Addae, E., & Boateng, R. (2018). Cloud computing research: A review of research themes, frameworks, methods and future research directions. *International Journal of Information Management, 38*(1), 128-139.

Sharma, S., & Saini, H. (2019). A novel four-tier architecture for delay aware scheduling and load balancing in fog environment. *Sustainable Computing: Informatics and Systems,, 24*. doi:https://doi.org/10.1016/j.suscom.2019.100355

Smajevic, M., & Bork, D. (2021). From conceptual models to knowledge graphs: a generic model transformation platform. *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion*, 610-614.

Thair, M., Ashraf, Q., & Dabbagh, M. (2019). Towards enabling autonomic computing in IoT ecosystem. *IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Techn*, 646-641.

Vailshery, L. S. (20 de Agosto de 2022). Recuperado el 20 de Septiembre de 2022, de Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2021, with forecasts from 2022 to 2030: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

Van Kranenburg, R., & Bassi, A. (2012). IoT challenges. *Communications in Mobile Computing*, 1-5.

Wang, A. (2020). Monitoring and Tracing @Netflix Streaming Data Infrastructure. *Software is changing the world Conference.* San Francisco. Obtenido de https://www.infoq.com/presentations/netflix-streaming-data-infrastructure/