



**Facultad de Ciencias de la Administración**

**Escuela de Ingeniería en Ciencias de la  
Computación**

**PROTOTIPO DE UNA APLICACIÓN MÓVIL  
BASADA EN MODELOS EN TIEMPO DE  
EJECUCIÓN PARA EL DESCUBRIMIENTO DE  
ENTIDADES IOT FÍSICAS Y DIGITALES**

**Trabajo de titulación previo a la obtención del grado  
de Ingeniero en Ciencias de la Computación**

**Autor**

Anthony Xavier Romero González

**Director**

Mgst. Fabián Marcelo Carvajal Vargas

**Cuenca – Ecuador**

**Año**

2024

## **AGRADECIMIENTO**

Quiero expresar mi profunda gratitud a mis padres, por su amor incondicional y su apoyo constante; a mi camarada, Patricio Ramón, compa de todo y pieza clave en el desarrollo del middleware de este prototipo; y a mi hermano Joel , por su gesto tan significativo de prestarme su teléfono. A mi tutora, Inés Acosta, agradezco su paciencia, guía y comprensión en cada etapa de este proyecto. Extiendo también mis agradecimientos a mis docentes tutores por su apoyo, a Territorios Inteligentes por brindarme una valiosa experiencia en desarrollo de software, y a la comunidad open source, cuyo compromiso ha puesto herramientas de calidad al alcance de todos, permitiendo también hacer realidad este prototipo.

## Índice de Contenidos

AGRADECIMIENTO .....	i
Índice de Contenidos.....	ii
Índice de figuras.....	iii
Índice de tablas .....	iv
RESUMEN .....	v
ABSTRACT.....	v
PROTOTIPO DE UNA APLICACIÓN MÓVIL BASADA EN MODELOS EN TIEMPO DE EJECUCIÓN PARA EL DESCUBRIMIENTO DE ENTIDADES IOT FÍSICAS Y DIGITALES.....	1
1. Introducción .....	1
1.1 Objetivos .....	1
1.1.1 Objetivo general.....	1
1.1.2 Objetivos específicos .....	1
1.2 Marco teórico .....	1
1.2.1 Internet de las cosas .....	1
1.2.2 Ingeniería dirigida por modelos .....	2
1.2.3 Models@runtime .....	3
1.2.4 m-DNS .....	3
2. Revisión de literatura .....	4
3. Materiales y métodos .....	5
3.1 Metodología .....	5
3.2 Herramientas .....	7
4. Aplicación móvil de descubrimiento de entidades IoT basada en modelos en tiempo de ejecución. ....	8
4.1 Diagrama de contexto de sistema.....	8
4.2 Arquitectura aplicación.....	9
4.3 Implementación de componentes.....	10
4.3.1 Capa de presentación .....	10
4.3.2 Capa de Dominio .....	13
4.3.3 Capa de Datos .....	15
5. Middleware de automodelado de arquitecturas de IoT .....	16
6. Estudio de caso .....	17
6.1 Escenario ilustrativo de IoT .....	17
6.2 Instanciación y evaluación empírica de la solución tecnológica.....	19
6.2.1 Configuración dispositivo IoT .....	19
6.2.2 Configuración dispositivo Bluetooth .....	21
6.2.3 Parámetros de Evaluación Empírica .....	22
6.2.4 Ejecución evaluación empírica .....	23
7. Conclusiones .....	28
8. Recomendaciones .....	28
9. Referencias.....	28

## Índice de figuras

<b>Figura 1</b> Metodología de investigación de Gorschek et al. (2006) adaptada al trabajo de titulación .....	6
<b>Figura 2</b> Diagrama de contexto de sistema referente a la aplicación móvil.....	8
<b>Figura 3</b> Arquitectura de aplicación móvil en base al patrón Clean Architecture. ....	9
<b>Figura 4</b> Pantalla inicio de sesión .....	10
<b>Figura 5</b> Pantalla autorización de permisos .....	11
<b>Figura 6</b> Pantalla listado de servicios descubiertos.....	11
<b>Figura 7</b> Pantalla interacción de servicios .....	12
<b>Figura 8</b> Fragmento entidad MdnsService.dart.....	13
<b>Figura 9</b> Fragmento entidad TxtProperties.dart .....	13
<b>Figura 10</b> Fragmento entidad MdnsServiceContext.dart .....	14
<b>Figura 11</b> Fragmento entidad MDnsServiceRepositoryImpl.dart.....	15
<b>Figura 12</b> Plano piso 4 de la facultad de filosofía.....	17
<b>Figura 13</b> Distribución de dispositivos .....	18
<b>Figura 14</b> Fragmento archivo de configuración publicación de servicio mdns. ....	20
<b>Figura 15</b> Comandos de inicialización servicio Avahi. ....	20
<b>Figura 16</b> Diagrama de componentes en la red perteneciente al estudio de caso. ....	21
<b>Figura 17</b> Fragmento archivo de configuración esp32 como dispositivo bluetooth .....	22
<b>Figura 18</b> Verificación e instanciación operativa de los dispositivos esp32 como dispositivos bluetooth .....	22
<b>Figura 19</b> Vistas de acceso y permiso.....	24
<b>Figura 20</b> Vistas de carga.....	24
<b>Figura 21</b> Vistas resultado de servicios disponibles .....	25
<b>Figura 22</b> Vistas escaneo de dispositivos BLE .....	25
<b>Figura 23</b> Vistas escaneo de dispositivos BLE .....	26

## Índice de tablas

<b>Tabla 1</b>	Lista de equipamientos empleados en el caso de estudio.....	19
<b>Tabla 2</b>	Especificación de los parámetros de evaluación empírica .....	23
<b>Tabla 3</b>	Resultado de evaluación empírica.....	27

# **PROTOTIPO DE UNA APLICACIÓN MÓVIL BASADA EN MODELOS EN TIEMPO DE EJECUCIÓN PARA EL DESCUBRIMIENTO DE ENTIDADES IOT FÍSICAS Y DIGITALES**

## **RESUMEN**

Este estudio desarrolló un prototipo de una aplicación móvil basado en modelos de tiempo de ejecución para el descubrimiento de entidades físicas y digitales de IoT dentro de la Universidad del Azuay. Basado en una arquitectura basada en eventos, la investigación integró dispositivos Bluetooth Low Energy (BLE) (ESP32) y servicios mDNS utilizando Avahi en máquinas virtuales que ejecutan Ubuntu 20.2, simulando sensores de IoT como detectores de temperatura y humo. Se emplearon FastAPI y Uvicorn para implementar servicios RESTful, facilitando la interacción entre la aplicación móvil y los dispositivos IoT. La evaluación empírica se realizó a través de un estudio de caso en tres ubicaciones distintas: Laboratorio 1, el pasillo y Laboratorio 3. La aplicación demostró con éxito la localización como el descubrimiento de servicios en entornos equipados con dispositivos BLE. Además, la interacción con los servicios RESTful resultó operativa y la interfaz de usuario se actualizó de manera eficiente en función de los resultados del escaneo en tiempo real. El sistema exhibió una estabilidad en diferentes ubicaciones, lo que validó la efectividad de la arquitectura basada en eventos para administrar y descubrir servicios de IoT. Los hallazgos indican que la solución desarrollada cumple con los objetivos de descubrir y gestionar con precisión las entidades de IoT, lo que proporciona una herramienta para la gestión de servicios de IoT en entornos académicos del mundo real.

**Palabras clave:** aplicaciones móviles BLE, computación consciente del contexto, Descubrimiento de servicios IoT, mDNS, Modelos en tiempo de ejecución

## **PROTOTYPE OF A RUN-TIME MODEL-BASED MOBILE APPLICATION FOR THE DISCOVERY OF PHYSICAL AND DIGITAL IOT ENTITIES**

### **ABSTRACT**

This study developed a prototype of a mobile application based on runtime models for the discovery of physical and digital IoT entities within the University of Azuay. Based on an event-driven architecture, the research integrated Bluetooth Low Energy (BLE) devices (ESP32) and mDNS services using Avahi on virtual machines running Ubuntu 20.2, simulating IoT sensors such as temperature and smoke detectors. FastAPI and Uvicorn were employed to implement RESTful services, facilitating the interaction between the mobile application and IoT devices. Empirical evaluation was performed through a case study in three distinct locations: Lab 1, the hallway, and Lab 3. The application successfully demonstrated localization as well as service discovery in environments equipped with BLE devices. Furthermore, interaction with RESTful services was operational and the user interface was efficiently updated based on real-time scanning results. The system exhibited stability across different locations, validating the effectiveness of the event-driven architecture for managing and discovering IoT services. The findings indicate that the developed solution meets the objectives of accurately discovering and managing IoT entities, providing a tool for IoT service management in real-world academic environments.

**Keywords:** BLE mobile applications, context-aware computing, IoT Service Discovery, mDNS, runtime models.

# **PROTOTIPO DE UNA APLICACIÓN MÓVIL BASADA EN MODELOS EN TIEMPO DE EJECUCIÓN PARA EL DESCUBRIMIENTO DE ENTIDADES IOT FÍSICAS Y DIGITALES**

## **1. Introducción**

En la era actual del Internet de las Cosas (IoT), la creciente complejidad y expansión de redes de dispositivos interconectados plantean desafíos críticos en términos de adaptabilidad e interoperabilidad. La investigación que se presenta es crucial debido a la necesidad imperante de desarrollar sistemas IoT que no solo gestionen eficazmente la heterogeneidad de dispositivos físicos y digitales, sino que también se adapten dinámicamente a cambios en el entorno y las necesidades específicas de los usuarios. Este estudio aborda directamente la problemática de crear un marco arquitectónico que facilite la interconexión y comunicación entre dispositivos heterogéneos, permitiendo la adaptación en tiempo real y el descubrimiento eficiente de entidades IoT relevantes basadas en la ubicación y el contexto del usuario.

La motivación detrás de esta investigación surge de la observación de que los sistemas actuales de IoT a menudo carecen de la flexibilidad necesaria para adaptarse a las condiciones cambiantes sin intervención manual extensiva, limitando su eficacia y eficiencia. La capacidad de un sistema para reconfigurarse automáticamente y descubrir nuevos servicios de manera transparente es esencial para el futuro de las tecnologías inteligentes y autónomas. Por lo tanto, esta investigación no solo es importante, sino fundamental para el avance de las tecnologías IoT, con el objetivo de mejorar significativamente la calidad de vida y la eficiencia operativa en diversos sectores aplicativos.

### **1.1 Objetivos**

#### **1.1.1 Objetivo general**

Desarrollar un prototipo de una aplicación móvil basada en modelos en tiempo de ejecución para el descubrimiento de entidades IoT físicas y digitales.

#### **1.1.2 Objetivos específicos**

- a) Realizar una indagación bibliográfica de la literatura relevante para identificar técnicas de posicionamiento en interiores, exteriores, así como el descubrimiento de servicios proporcionados por entidades IoT.
- b) Analizar, diseñar y codificar un prototipo de aplicación móvil basada en modelos de tiempo de ejecución para descubrir entidades IoT.
- c) Analizar, diseñar y codificar un middleware de automodelado de arquitecturas de IoT.
- d) Evaluar empíricamente la solución tecnológica (aplicación móvil y middleware) mediante un estudio de caso.

### **1.2 Marco teórico**

#### **1.2.1 Internet de las cosas**

Kevin Ashton fue el pionero en proponer el concepto del Internet de las Cosas (IoT) en 1999, describiéndolo como objetos conectados e interoperables, identificables de manera única y equipados con tecnología de identificación por radiofrecuencia (RFID) (Li et al., 2015). Desde entonces, el IoT ha evolucionado dramáticamente, convirtiéndose en una red global de dispositivos interconectados capaces de recoger, comunicar e intercambiar datos acerca de su entorno inmediato. Hoy día, se cuentan miles de millones de estos dispositivos, que si bien han dependido tradicionalmente de arquitecturas basadas en la nube para el procesamiento y almacenamiento de datos, enfrentan retos significativos relacionados con el ancho de banda y los retrasos en la comunicación. Esta situación ha incentivado el desarrollo de soluciones de computación en el borde y en la niebla (Edge and Fog computing), las cuales buscan descentralizar estas funciones y ubicarlas más cerca de las fuentes de datos, mejorando así la eficiencia y la respuesta de los dispositivos (Alfonso et al., 2023).

La Internet de las Cosas (IoT) comprende diversidad de tecnologías y componentes que deben operar conjuntamente en un sistema dinámico y adaptable a las necesidades específicas de sus usuarios. Según Bonetto et al. (2012), la omnipresencia de las fuentes de información en el IoT implica que se generará, transmitirá, recogerá, almacenará y procesará una gran cantidad de datos sobre casi todos los aspectos de la actividad humana, tanto pública como privada. Esta naturaleza intrínseca del IoT destaca la capacidad de producir y gestionar volúmenes masivos de datos que informan y forman la base para decisiones en tiempo real en diversos contextos. Por otro lado, Hendriks (2016) enfatiza que el IoT no debe considerarse un sistema integrado per se, sino más bien como una colección de componentes derivados de diversas industrias que deben vincularse según el contexto. Esta necesidad subraya la importancia de mantener una apertura funcional en los sistemas IoT, al permitir que los usuarios personalicen el sistema según sus necesidades individuales. La estandarización abierta permite la interoperabilidad entre los diferentes componentes del IoT y fomenta la innovación a través de normas técnicas que pueden ser establecidas de múltiples maneras.

Además, Ray (2018) describe la arquitectura del IoT como un sistema que puede ser físico, virtual o una combinación de ambos, compuesto por una colección de numerosos objetos físicos activos, sensores, actuadores, servicios en la nube, protocolos específicos del IoT, capas de comunicación, usuarios, desarrolladores y una capa empresarial. Esta arquitectura actúa como un componente central de la infraestructura específica del IoT, facilitando un enfoque sistemático hacia los componentes dispares y proporcionando soluciones a problemas relacionados. Por último, Vermesan et al. (2022) conceptualizan el IoT como una parte integrante del Futuro Internet, definido como una infraestructura de red global dinámica con capacidades de autoconfiguración basadas en protocolos de comunicación estándar e interoperables, donde los objetos físicos y virtuales tienen identidades, atributos físicos, personalidades virtuales, utilizan interfaces inteligentes y están integrados de manera transparente en la red de información.

### **1.2.2 Ingeniería dirigida por modelos**

La creciente complejidad y la omnipresencia del software en la sociedad actual han experimentado un aumento exponencial. Como señalan Hutchinson et al. (2011), la única manera viable de gestionar esta complejidad y continuar brindando los beneficios del software al público es mediante el uso de métodos de abstracción adecuados, lo que implica la utilización sistemática de modelos como artefactos primarios en el proceso de ingeniería de software. En relación con esta idea, Khalil & Dingel (2018) sugieren que la ingeniería de software dirigida por modelos (MDE) se centra en mejorar la productividad y la calidad de los artefactos de software al priorizar los modelos como elementos fundamentales, relegando el código a un segundo plano. Este enfoque resalta la importancia de los modelos en la simplificación y eficiencia del desarrollo de software, enfatizando su valor en comparación con la codificación tradicional.

Bézivin (2004) amplía esta perspectiva introduciendo la Arquitectura Dirigida por Modelos (MDA), una variante específica dentro de una tendencia global más amplia conocida como ingeniería de modelos. Las ideas fundamentales de la ingeniería de modelos son aplicables a diversas aproximaciones, como la programación generativa, los lenguajes específicos de dominio, la computación integrada en modelos y las fábricas de software. MDA se define como la implementación de los principios de la ingeniería de modelos en torno a un conjunto de estándares de la OMG, como MOF (Meta Object Facility), XMI (XML Metadata Interchange), OCL (Object Constraint Language), UML (Unified Modeling Language), CWM (Common Warehouse Metamodel) y SPEM (Software Process Engineering Metamodel), entre otros. Este enfoque reitera el principio básico de que "todo es un objeto", un concepto fundamental en la programación orientada a objetos que fue crucial en la década de 1980 para sentar las bases de esta disciplina.

Según Fondement & Silaghi (2004) la Ingeniería Dirigida por Modelos (MDE) se define como un enfoque en la ingeniería de software que organiza los niveles de abstracción y las metodologías, alentando a los desarrolladores a utilizar modelos para describir tanto el problema como la solución en diferentes niveles de abstracción. MDE proporciona un marco que permite a los metodólogos definir qué modelo utilizar en cada momento y cómo reducir el nivel de abstracción al establecer las relaciones entre los modelos implicados. Este enfoque facilita la clara definición de metodologías, el desarrollo de sistemas en cualquier nivel de abstracción, y la organización y automatización de las actividades de prueba y validación.

En un proceso MDE, es fundamental que se definan los siguientes aspectos:



1. Cantidad de niveles de abstracción: Cuántos niveles existen y qué plataformas deben integrarse.
2. Notaciones de modelado y sintaxis abstracta: Qué notaciones se utilizarán en cada nivel de abstracción.
3. Refinamientos: Cómo se realizan los refinamientos y qué plataformas e información adicional se integran en los niveles inferiores de abstracción.
4. Generación y despliegue de código: Cómo se genera el código para el lenguaje de modelado en el nivel más bajo de abstracción y cómo se despliega.
5. Verificación y validación: Cómo se verifica un modelo respecto al modelo del nivel superior, cómo se valida, y cómo se generan casos de prueba para el sistema en desarrollo.

### 1.2.3 Models@runtime

En el ámbito de los sistemas de software temporales de misión crítica, es crucial que estos sistemas puedan adaptarse de manera segura a los cambios en su entorno de ejecución sin necesidad de ser desconectados, debido a las funciones vitales que desempeñan (Blair et al., 2009). Estos sistemas deben, idealmente, modificar su comportamiento en tiempo de ejecución con mínima o ninguna intervención humana. Para facilitar este tipo de adaptación autónoma, se utilizan los modelos en tiempo de ejecución, conocidos como *model@runtime*. Estos modelos representan una autorrepresentación del sistema que está causalmente conectada con él, resaltando la estructura, el comportamiento o los objetivos del sistema desde una perspectiva orientada al espacio del problema.

Un modelo es una abstracción de algún aspecto de un sistema, y que el sistema descrito puede o no existir en el momento de la creación del modelo. Los modelos se desarrollan para cumplir propósitos específicos, como proporcionar una descripción comprensible para los humanos o presentar información en un formato que pueda ser analizado mecánicamente. En MDE, los modelos no solo son los artefactos primarios del desarrollo, sino también el medio principal por el cual los desarrolladores y otros sistemas entienden, interactúan, configuran y modifican el comportamiento en tiempo de ejecución del software. (France & Rumpe, 2007)

Además, Bencomo et al. (2019) establecen que los modelos en tiempo de ejecución representan una capa de reflexión que está causalmente conectada con el sistema subyacente, de modo que cada cambio en el modelo de tiempo de ejecución provoca un cambio en el sistema reflejado y viceversa. *Models@runtime* combina los principios de la reflexión computacional con la ingeniería dirigida por modelos, y se puede utilizar como un catalizador para la creación de software futuro que requiere ser duradero mientras enfrenta condiciones ambientales en constante cambio, las cuales solo se conocen parcialmente en la fase de desarrollo. Bouhamed et al. (2021) explican que la visión de *models@runtime* implica el uso de modelos no solo en la fase de diseño, sino también durante la ejecución. Los sistemas subyacentes y sus modelos correspondientes evolucionan juntos y se influyen mutuamente durante la ejecución de estos sistemas. Este paradigma permite que los sistemas en ejecución enfrenen los cambios dinámicos del entorno y satisfagan los requisitos de los usuarios.

### 1.2.4 m-DNS

En el ámbito del Internet de las Cosas (IoT), donde se anticipa la interconexión de un gran número de dispositivos con recursos limitados, se requiere soluciones eficientes y simplificadas para el descubrimiento tanto de dispositivos como de servicios. Según Siljanovski et al. (2014), la adopción de soluciones estandarizadas y ampliamente reconocidas, como Multicast DNS (mDNS) en combinación con DNS Service Discovery (DNS-SD), permite una integración más rápida con la infraestructura existente de Internet, aprovechando las herramientas y sistemas ya disponibles. Este enfoque ofrece una solución que cumple con los requisitos de eficiencia y simplicidad, facilitando el descubrimiento de servicios en redes locales sin necesidad de configuraciones complejas.

La relevancia de mDNS radica en su capacidad para realizar operaciones similares a DNS en un entorno de enlace local sin la dependencia de un servidor DNS unicast convencional. Como destacan (Chesire & Krochmal (2013), mDNS permite a los dispositivos en red buscar y resolver registros de recursos DNS, incluyendo nombres de host, sin la necesidad de infraestructura DNS tradicional. Además, mDNS asigna una porción del espacio de nombres DNS para uso local, eliminando la necesidad de tarifas anuales, delegaciones o configuraciones adicionales. Los principales beneficios de

mDNS incluyen la escasa o nula necesidad de administración, su capacidad para operar en ausencia de infraestructura, y su resiliencia durante fallos de infraestructura.

Por otra parte, la técnica de Multicast DNS Service Discovery (mDNS-SD) ha ganado prominencia como un método efectivo para la distribución de servicios en redes locales sin requerir configuración previa. Kaiser & Waldvogel (2014), explican que mDNS-SD se basa en las dos capas superiores de la pila Zeroconf, donde DNS Service Discovery se construye sobre Multicast DNS. Al estar basada en multicast, cada dispositivo en la red recibe automáticamente todo el tráfico de anuncios, obteniendo información sobre los usuarios y servicios en la red sin necesidad de transmitir paquetes adicionales. A través de mDNS-SD, los dispositivos publican sus nombres de host, que a menudo incluyen el nombre del usuario, junto con información sobre los servicios ofrecidos y solicitados al ingresar a una red, facilitando una interacción ágil y sin fricciones en entornos locales.

## 2. Revisión de literatura

En el ámbito del Internet de las Cosas (IoT), los esfuerzos de investigación y desarrollo se han centrado en generar soluciones que armonicen las tecnologías emergentes con las dinámicas necesidades de los usuarios y los entornos variables. Las indagaciones en este campo comienzan con la formulación de arquitecturas y plataformas que se alinean con tendencias tecnológicas y científicas clave, como la Web de las Cosas (Martín et al., 2016). Esta plataforma no solo facilita interacciones restful con el contexto a través del uso de estándares web para la conexión de dispositivos embebidos, sino que también incorpora tecnologías como iBeacons para superar las restricciones de localización en interiores, representando un progreso significativo en la interacción entre dispositivos inteligentes y usuarios.

Continuando con este enfoque, la arquitectura desarrollada por Juyoung et al. (2013) se centra específicamente en superar las restricciones en el descubrimiento y la composición de servicios que presentan numerosas arquitecturas de middleware anteriores, a través de la implementación de servicios web semánticos. Este método no solo facilita una mejor integración de los servicios, sino que también hace una clara distinción entre los servicios que son conscientes del contexto y la información contextual misma, estableciendo así un fundamento sólido para el desarrollo de sistemas que son tanto más inteligentes como más adaptables.

La importancia de la adaptabilidad y la personalización en el diseño de sistemas IoT se ve reforzada por las contribuciones de Gomez-Torres & Lujan-Mora (2017), quienes enfatizan la implementación de un Servicio de Monitoreo de Contexto utilizando una Arquitectura Orientada a Servicios (SOA). Esta propuesta subraya la necesidad de monitorear las variaciones de contexto y ajustar los servicios en consecuencia, un principio que se extiende al diseño de sistemas distribuidos y dinámicos. Chen et al. (2016) ilustra cómo la integración de tecnología móvil con escenarios auténticos, mediante el uso de tecnología iBeacon en un museo de ciencias, no solo mejora la experiencia de aprendizaje de los visitantes, sino que también inspira su entusiasmo. Esta convergencia de tecnología, contexto y contenido educativo resalta la capacidad transformadora de las soluciones IoT bien diseñadas.

Wei & Chan (2013) presentan CAMPUS, un middleware orientado al contexto para servicios ubicuos, diseñado para la toma de decisiones automatizadas en tiempo de ejecución. CAMPUS se basa en tres enfoques técnicos principales: ontologías, razonamiento descriptivo y adaptación composicional. A diferencia de otros middlewares que dependen de decisiones predefinidas, CAMPUS se adapta dinámicamente a los cambios contextuales. Implementa un modelo de decisión de múltiples etapas, que filtra, selecciona y preprocesa para elegir la mejor alternativa en una tarea específica. Estas decisiones se toman en la capa de decisión y se envían a la capa de programación, que reconfigura aplicaciones contextuales. La capa de conocimiento, compuesta por ontologías de Servicio, Contexto y Tasklet, es esencial para las decisiones adaptativas. CAMPUS, como middleware basado en semántica, permite decisiones de adaptación en tiempo de ejecución, libera a los desarrolladores de la intervención manual.

La utilidad de estas arquitecturas se evidencia en aplicaciones específicas, como las que presentan Shapsough & Zualkernan (2020), resaltan la utilidad de emplear dispositivos IoT de borde en contextos de aprendizaje ubicuo, resaltando la eficacia del CoAP y las capacidades del MQTT para comunicaciones de publicación/suscripción. Por otro lado, Sykes (2020) ilustra el uso de balizas Bluetooth Low Energy en la monitorización de seguridad interna, mostrando la eficacia de soluciones contextuales en ambientes complejos.

K. Khalil et al. (2020) destaca que los enfoques de descubrimiento dependen de varios parámetros, como la gestión de configuración, la inscripción y desinscripción de dispositivos, o los estados de reposo de los objetos físicos. Los métodos de descubrimiento consideran parámetros adicionales como la ubicación, el contexto y el control de acceso basado en humanos, entre otros factores relevantes. En un enfoque complementario, Murturi et al. (2019) proponen un marco basado en semántica que utiliza el mecanismo de publicidad de servicios para objetos inteligentes. Esta publicidad de servicio incluye metadatos, tales como nombre, identificador, ubicación, punto de conexión y un enlace a la anotación semántica.

Liao & Weng (2023) abordan una limitación presente en la mayoría de los mecanismos de descubrimiento de servicios existentes, incluido mDNS/DNS-SD, que es la ausencia de información espacial. Si bien mDNS/DNS-SD permite descubrir dispositivos IoT conectados por nombre y tipo de servicio, y realizar selecciones sofisticadas mediante la adición de filtros como pares clave-valor en los registros de recursos DNS, carece de la capacidad para integrar directamente información sobre espacio y ubicación. En su estudio, Liao y Weng presentan un enfoque innovador para integrar de manera fluida la información espacial y de ubicación en mDNS/DNS-SD. Su trabajo incluye la especificación formal de este enfoque, el desarrollo de un prototipo, y experimentos que demuestran la eficacia y el potencial de la metodología propuesta, ampliando significativamente las capacidades de los sistemas de descubrimiento de servicios en entornos IoT.

Stolikj et al. (2016) propone una extensión al protocolo mDNS/DNS-SD para considerar el contexto de los servicios mediante el uso de etiquetas que permiten una identificación de los servicios. Estas etiquetas actúan como descriptores atómicos de atributos específicos del servicio, como la ubicación o las capacidades del dispositivo, y pueden ser preconfiguradas durante la fase de despliegue del servicio o aprendidas dinámicamente en tiempo de ejecución. El proceso de descubrimiento de servicios comienza cuando el cliente emite una consulta que incluye una combinación de etiquetas de contexto deseadas o no deseadas, estructuradas mediante operadores lógicos booleanos, lo que permite expresar condiciones contextuales complejas. Una vez recibida la consulta, el protocolo evalúa cada instancia de servicio en función de las etiquetas de contexto especificadas, devolviendo al cliente solo aquellas que cumplen con los criterios contextuales. Las etiquetas se almacenan en una forma compacta, optimizando el uso de memoria en comparación con métodos tradicionales que dependen de nombres de dominio completamente calificados o pares clave-valor. Este enfoque reduce el volumen de consultas al permitir un filtrado más preciso, disminuyendo el número de respuestas irrelevantes en redes grandes. Además, la extensión está diseñada para ser compatible con las versiones anteriores de los estándares mDNS/DNS-SD, lo que facilita su integración en sistemas actuales sin requerir cambios significativos en la infraestructura existente.

En cuanto respecta a descubrimiento de balizas, iBeacon, desarrollado por Apple, es una herramienta ampliamente utilizada en dispositivos Apple para el reconocimiento de ubicación basada en la proximidad del emisor de la baliza, operando mediante el protocolo Bluetooth Low Energy (BLE). Sin embargo, su funcionalidad se limita a la publicidad unidireccional y requiere la preasignación de UUIDs, con dispositivos que permanecen activos en modo de publicidad de forma continua. En un esfuerzo por mejorar las capacidades del BLE en aplicaciones IoT, Hirsch et al. (2023) introducen DynGATT, un protocolo que extiende el perfil GATT de BLE para soportar estructuras IoT dinámicas, combinando las ventajas de IPv6 y GATT.

### **3. Materiales y métodos**

#### **3.1 Metodología**

La principal dificultad al desarrollar un sistema IoT que sea autoadaptable y auto modelable reside en establecer un marco que no solo mejore la interconexión y la comunicación entre un diverso conjunto de dispositivos físicos y digitales, sino que también permita adaptaciones en tiempo real ante los cambios dinámicos en el entorno y las demandas del usuario. Esto implica la identificación eficaz de entidades IoT pertinentes basadas en la ubicación y el contexto del usuario, haciendo uso de tecnologías como Bluetooth, WiFi y GPS. La solución desarrollada debe manejar la diversidad de dispositivos, garantizando tanto su interoperabilidad como su capacidad para reconfigurarse o identificar nuevos servicios de manera transparente.

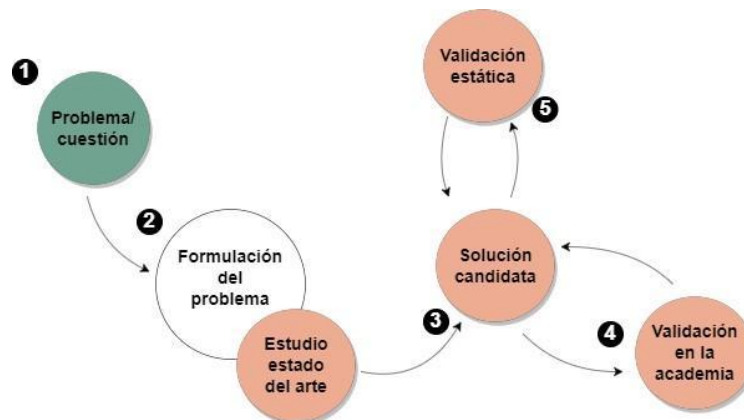
En este estudio, se utilizan modelos en tiempo de ejecución para proporcionar una base semántica que fundamenta la toma de decisiones relacionadas con la adaptación del sistema en tiempo real, contemplando además aspectos cruciales como la verificación y el monitoreo. Esta metodología

modifica la funcionalidad de la aplicación móvil, que se rige según las reglas definidas en tiempo de ejecución para descubrir e interactuar con componentes IoT. Mediante estos modelos, la aplicación no solo identifica automáticamente los componentes IoT relevantes en su entorno, sino que también ajusta su comportamiento y configuraciones de acuerdo con las reglas del modelo, asegurando así interacciones coherentes y contextualmente adecuadas con estos dispositivos. Esto facilita un ecosistema IoT más integrado y reactivo, donde la aplicación móvil opera de manera dinámica y adaptativa, mejorando la experiencia del usuario y la eficiencia operativa del sistema.

La metodología que se aplica en este trabajo de titulación se fundamenta en el Modelo de Transferencia Tecnológica propuesto por Gorschek et al. (2006), adaptando cinco de sus siete etapas originales para alinearlas con los objetivos específicos del proyecto. Las etapas seleccionadas para implementar son las siguientes:

**Figura 1**

*Metodología de investigación de Gorschek et al. (2006) adaptada al trabajo de titulación*



- 1) **Identificación del problema:** El problema central se define como la necesidad de descubrir servicios RESTful a través de agentes de descubrimiento basados en teléfonos celulares. Este desafío implica gestionar la publicación y el descubrimiento de servicios en una red, así como capturar y administrar la información contextual.
- 2) **Formulación del Problema:** Basado en la problemática identificada, se establece como objetivo desarrollar un marco que permita descubrir y gestionar servicios RESTful en dispositivos móviles, utilizando mDNS para la detección de servicios. La solución debe abordar la interoperabilidad entre dispositivos, la gestión eficiente de la información contextual, y la capacidad de operar tanto en redes conectadas como desconectadas. Se lleva a cabo una revisión de la literatura existente para identificar herramientas, protocolos, y enfoques actuales en el descubrimiento de servicios, el uso de modelos en tiempo de ejecución, y la gestión del contexto. Este análisis proporcionará la base teórica necesaria para desarrollar el prototipo de la aplicación y garantizar que se apliquen las mejores prácticas y tecnologías disponibles.
- 3) **Desarrollo de la Solución Candidata:** Se propone la creación de un prototipo de aplicación móvil utilizando un framework multiplataforma, en el cual se implementará el agente de descubrimiento. La aplicación utilizará mDNS para encontrar servicios publicados en máquinas virtuales que anuncian servicios mediante el mismo protocolo. Además, se establecerá una conexión a un servidor centralizado que contiene los modelos en tiempo de ejecución y gestiona la información contextual capturada por el teléfono, devolviendo los servicios autorizados para cada contexto.
- 4) **Validación en la Academia:** La solución candidata será presentada y discutida con docentes tutores para obtener retroalimentación crítica y realizar los ajustes necesarios. Este proceso incluirá discusiones en profundidad para evaluar la aceptación y aplicabilidad de la solución propuesta.
- 5) **Validación estática:** La solución candidata será sometida a un estudio de caso, en el cual se evaluará su conformidad con los lineamientos y procesos propuestos por Wohlin et al. (2012). Se llevarán a cabo pruebas iniciales en un entorno controlado para verificar el correcto funcionamiento de los mecanismos de descubrimiento y gestión de servicios.

### 3.2 Herramientas

Flutter es un kit de herramientas de interfaz de usuario portátil desarrollado por Google, diseñado para crear aplicaciones compiladas de forma nativa en móviles, web y escritorio a partir de una única base de código. Busca simplificar el desarrollo de aplicaciones al reducir los costos y la complejidad de producción a través de plataformas. Esto es posible gracias a su motor de renderizado que dibuja widgets de manera directa, sin depender de la tecnología del navegador ni de los widgets nativos del dispositivo. Flutter es gratuito y de código abierto, lo que ha llevado a su adopción global por parte de desarrolladores y organizaciones (Google, 2023).

Flutter permite el desarrollo ágil de aplicaciones, reduciendo la barrera de entrada para el desarrollo multiplataforma al ofrecer una base de código única para múltiples entornos. En el caso específico de Android, Flutter se integra de manera nativa mediante la compilación adelantada de código Dart en bibliotecas ARM y x86-64, lo que garantiza un rendimiento óptimo en dispositivos móviles Android. Además, Flutter aprovecha el uso del NDK (Native Development Kit) de Android, lo que permite acceder a características y capacidades nativas del sistema operativo, como la cámara, el GPS y la red, sin comprometer el rendimiento.

Para el desarrollo del proyecto de tesis, se han utilizado diversas librerías en Flutter que son fundamentales para la gestión de estados, persistencia de datos, descubrimiento de servicios mDNS, y escaneo de dispositivos Bluetooth. A continuación, se describen las principales librerías empleadas:

- `multicast_dns` : La librería `multicast_dns` es un paquete de Dart que permite realizar el descubrimiento de servicios mediante multicast DNS (mDNS), compatible con Bonjour y Avahi, y se basa en el estándar RFC 6762. Esta librería permite descubrir dispositivos y servicios en redes locales sin requerir un servidor DNS centralizado. Dentro del proyecto, su uso es clave para identificar y descubrir servicios disponibles en la red local, facilitando la interacción con dispositivos IoT de manera transparente. Su implementación está diseñada para ser utilizada tanto en Android como en otras plataformas compatibles con Flutter.
- `Sqflite`: La librería `sqflite` es un plugin de SQLite para Flutter que soporta iOS, Android y MacOS. Ofrece soporte para transacciones y lotes, gestión automática de versiones de la base de datos, y proporciona ayudas para realizar consultas de inserción, actualización, eliminación y selección. En este proyecto, `sqflite` se utiliza para persistir la información del usuario, así como los servicios mDNS descubiertos y el contexto de la aplicación. Además, ayuda a evitar la sobresaturación de peticiones al middleware de servicios, ya que permite realizar operaciones de almacenamiento y recuperación de datos localmente, reduciendo la dependencia de solicitudes HTTP constantes y mejorando la eficiencia global del sistema.
- `Dio`: `dio` es un paquete de Dart/Flutter que facilita la comunicación HTTP, ofreciendo configuraciones globales, interceptores, cancelación de solicitudes, subida/descarga de archivos, gestión de tiempo de espera, entre otras funcionalidades. En este proyecto, `dio` se utiliza para gestionar la comunicación mediante el protocolo HTTP, permitiendo el envío y recepción de datos como los servicios descubiertos, la información del usuario, y el contexto de la aplicación hacia y desde el middleware.
- `flutter_bloc`: La librería `flutter_bloc` facilita la implementación del patrón de diseño BLoC (Business Logic Component) en Flutter. Este patrón separa la lógica de negocio de la interfaz de usuario, permitiendo una mejor gestión del estado y de la arquitectura de la aplicación, siguiendo principios SOLID. En el proyecto, `flutter_bloc` es utilizado para gestionar la lógica de negocio y el estado de los diferentes componentes, lo que garantiza una estructura escalable y mantenible. Al estructurar la aplicación de esta manera, se asegura una clara separación de responsabilidades, mejorando tanto el desarrollo como el mantenimiento del código a largo plazo.
- `flutter_blue_plus`: La librería `flutter_blue_plus` permite la conexión y comunicación con dispositivos Bluetooth Low Energy (BLE) en Android, iOS y MacOS. En este proyecto, se utiliza para escanear dispositivos Bluetooth filtrados por su UUID, con el objetivo de obtener el nombre de los dispositivos autorizados más cercanos. Esta información es fundamental para determinar la localización física en tiempo real dentro del contexto de la aplicación. El nombre de los dispositivos Bluetooth descubiertos se envía al middleware, que utiliza esta información para ajustar la ubicación en tiempo real.
- `permission_handler`: `permission_handler` es una librería que proporciona una API multiplataforma para gestionar permisos en iOS y Android. Esta librería es esencial para gestionar las autorizaciones del dispositivo, como la localización y el uso de Bluetooth, necesarios para la correcta operación de la aplicación. En Android, requiere ajustes en los

archivos de manifiesto para garantizar que se soliciten y gestionen los permisos adecuados. Además, permite mostrar una justificación al usuario cuando se solicitan ciertos permisos, mejorando la experiencia y seguridad de la aplicación.

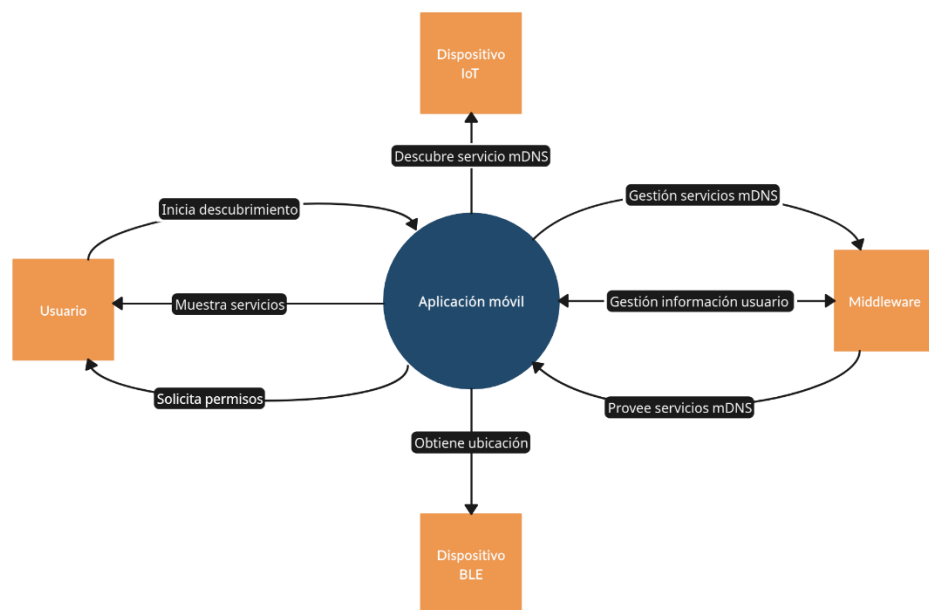
## 4. Aplicación móvil de descubrimiento de entidades IoT basada en modelos en tiempo de ejecución.

### 4.1 Diagrama de contexto de sistema

El Diagrama de Contexto del Sistema proporciona una visión general de las interacciones fundamentales entre los principales actores y componentes del sistema. En este caso, ilustra cómo el usuario interactúa con la aplicación móvil para descubrir y gestionar servicios IoT a través de la red mDNS y dispositivos Bluetooth. Además, la figura 2 muestra la comunicación entre la aplicación móvil y el middleware, que procesa y adapta los servicios descubiertos según el contexto del usuario.

**Figura 2**

*Diagrama de contexto de sistema referente a la aplicación móvil.*



- Usuario - Aplicación móvil (Inicia descubrimiento): El usuario interactúa directamente con la aplicación móvil para iniciar el proceso de descubrimiento de servicios disponibles en la red. A través de la interfaz de usuario, luego de iniciar sesión y autorizar los permisos de localización y Bluetooth, se habilita la funcionalidad de descubrimiento de servicios IoT y dispositivos Bluetooth cercanos.
- Aplicación móvil - Dispositivo IoT (Descubre servicio mDNS): La aplicación móvil, utilizando la librería `multicast_dns`, realiza el descubrimiento de servicios publicados por dispositivos IoT a través del protocolo mDNS. Estos servicios son detectados localmente en la red sin necesidad de servidores DNS centralizados. La información del servicio incluye el nombre del mismo, la dirección IP local, una lista de enlaces RESTful, y propiedades como localización y tipo de servicio.
- Aplicación móvil - Middleware (Gestión servicios mDNS): Una vez que la aplicación móvil detecta los servicios mDNS, estos son gestionados a través del middleware, que se encarga de procesar, organizar y proporcionar los servicios disponibles en función del contexto del usuario. Los servicios se almacenan tanto en el teléfono como en el middleware, lo que permite mantener actualizada la información sobre los servicios IoT descubiertos.
- Middleware - Aplicación móvil (Provee servicios mDNS): El middleware ofrece a la aplicación móvil los servicios mDNS detectados y gestionados, permitiendo que estos sean mostrados al usuario. Además, tiene en cuenta la ubicación enviada por el teléfono y los

permisos del usuario para devolver los servicios según el contexto y las autorizaciones otorgadas.

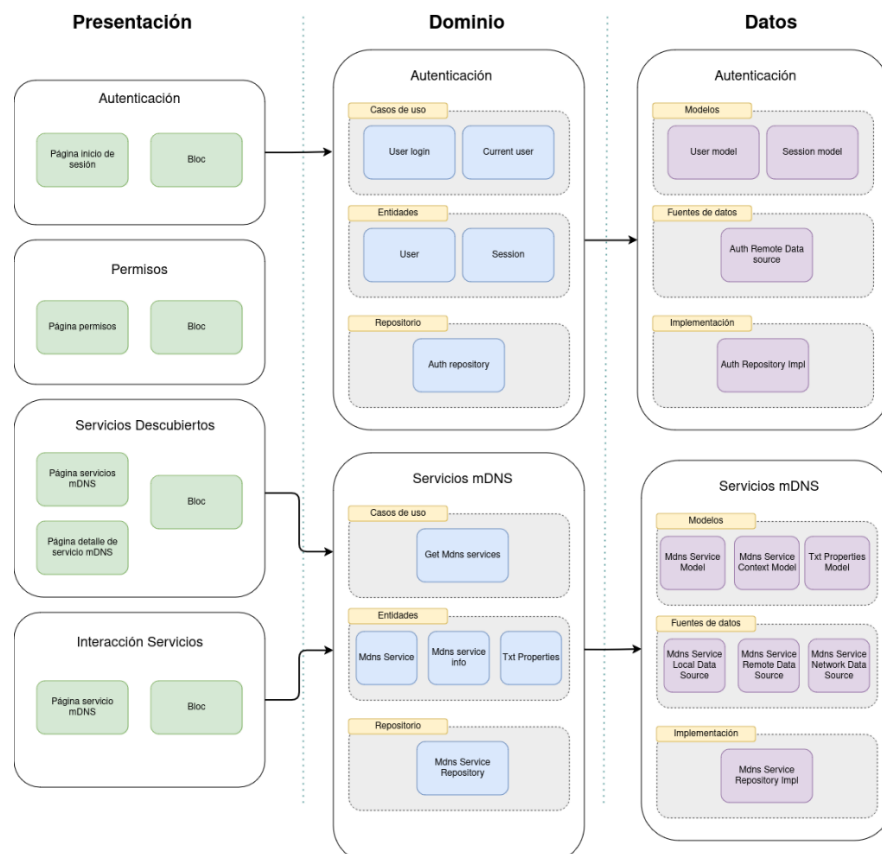
- Aplicación móvil - Middleware (Gestión información usuario): La aplicación móvil envía al middleware información sobre el usuario y su contexto, como preferencias o configuraciones, necesarias para personalizar los servicios IoT descubiertos, modificar la interfaz de la aplicación, y mejorar la interacción del usuario con su entorno.
- Aplicación móvil - Dispositivo BLE (Obtiene ubicación): Utilizando la librería flutter\_blue\_plus, la aplicación móvil realiza un escaneo de dispositivos Bluetooth Low Energy cercanos. Estos dispositivos, filtrados por su UUID, permiten a la aplicación obtener la ubicación física del usuario, que se representa a través de los nombres de los dispositivos BLE descubiertos.
- Usuario - Aplicación móvil (Solicita permisos): Antes de interactuar con los dispositivos BLE y otros servicios de la red, la aplicación móvil solicita los permisos necesarios al usuario, como el acceso a la localización y el uso de Bluetooth, mediante la librería permission\_handler. Esto garantiza que la aplicación cumpla con las políticas de seguridad y privacidad del dispositivo, y permita interactuar con los dispositivos IoT y BLE.
- Aplicación móvil - Usuario (Muestra servicios): La aplicación móvil muestra al usuario los servicios descubiertos a través de la interfaz. Esta información incluye los dispositivos IoT detectados, así como los servicios REST disponibles. El usuario puede interactuar con estos servicios y tomar decisiones informadas basadas en la información proporcionada.

## 4.2 Arquitectura aplicación

La aplicación móvil desarrollada para este proyecto sigue el patrón de diseño Clean Architecture representado en la figura 3, el cual promueve la separación de responsabilidades y la independencia entre las distintas capas del software. Este enfoque facilita la mantenibilidad, escalabilidad y testabilidad de la aplicación, aspectos cruciales en el desarrollo de sistemas complejos como los que integran dispositivos IoT y tecnologías de descubrimiento de servicios.

**Figura 3**

*Arquitectura de aplicación móvil en base al patrón Clean Architecture.*



El diagrama se divide en tres capas principales, cada una representando una parte fundamental de la aplicación:

- Capa de Presentación interactúa con la Capa de Dominio mediante los Casos de Uso. Los BloCs o Cubits en la capa de presentación envían eventos a los casos de uso para ejecutar la lógica de negocio.
- Capa de Dominio define las Entidades, Repositorios y Casos de Uso. Los repositorios son interfaces que definen contratos para obtener datos.
- Capa de Datos implementa los repositorios definidos en la capa de dominio. Contiene las Fuentes de Datos y los Modelos que representan cómo se almacenan y reciben los datos.

Para la Inversión de Dependencias se considera:

- La capa de dominio no debe depender de ninguna otra. Las capas externas dependen de la capa de dominio.
- Cada capa tiene responsabilidades claras y definidas, lo que facilita el mantenimiento y escalabilidad de la aplicación.
- Los repositorios en la capa de dominio son interfaces, y sus implementaciones se encuentran en la capa de datos.
- La información fluye desde la capa de datos hacia la capa de presentación a través de la capa de dominio.

## 4.3 Implementación de componentes

### 4.3.1 Capa de presentación

#### 4.3.1.1 Pagina inicio de sesión

**Figura 4**

*Pantalla inicio de sesión*



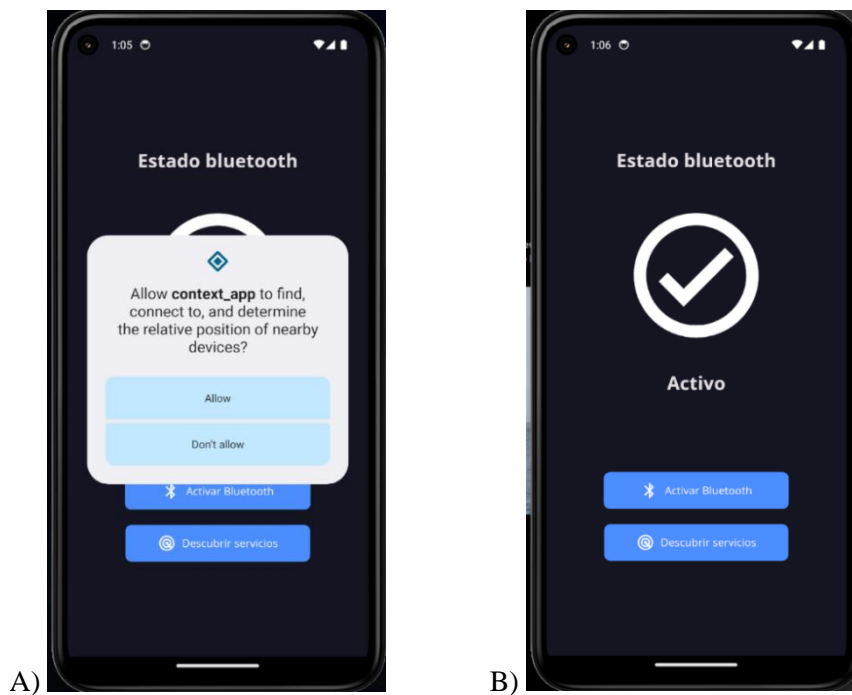
Permite al usuario ingresar sus credenciales y desencadena el proceso de autenticación mediante el componente AuthBloc (ver figura 4).



### 4.3.1.2 Pagina inicio de permisos

**Figura 5**

*Pantalla autorización de permisos*

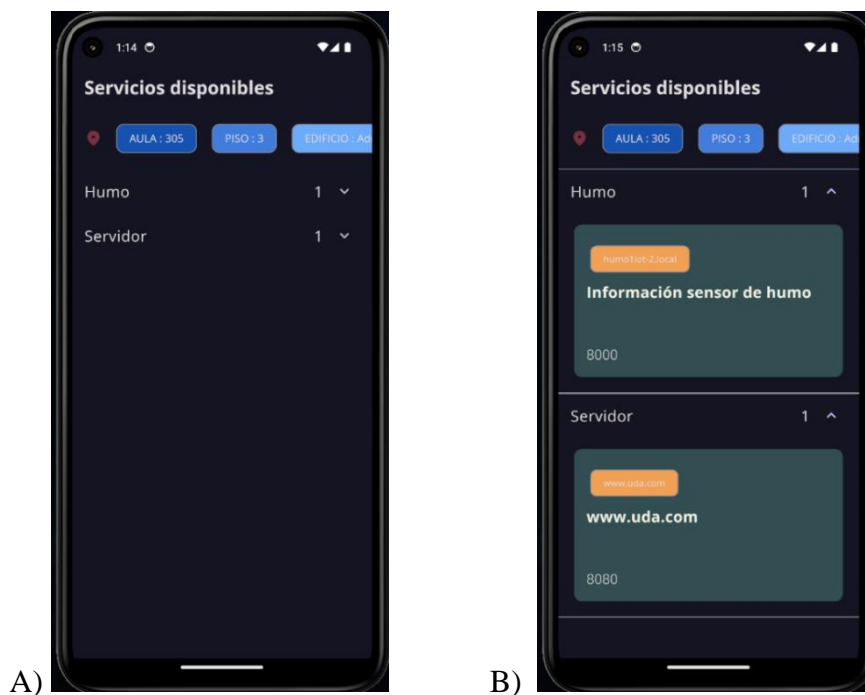


Permite al usuario autorizar a la aplicación hacer uso de la localización y bluetooth. Al ingresar en esta página se despliega el cuadro de dialogo del sistema por defecto para solicitar autorización para encontrar, conectar y determinar la posición relativa de dispositivos cercanos (ver figura 5).

### 4.3.1.3 Pagina servicios mDNS

**Figura 6**

*Pantalla listado de servicios descubiertos*



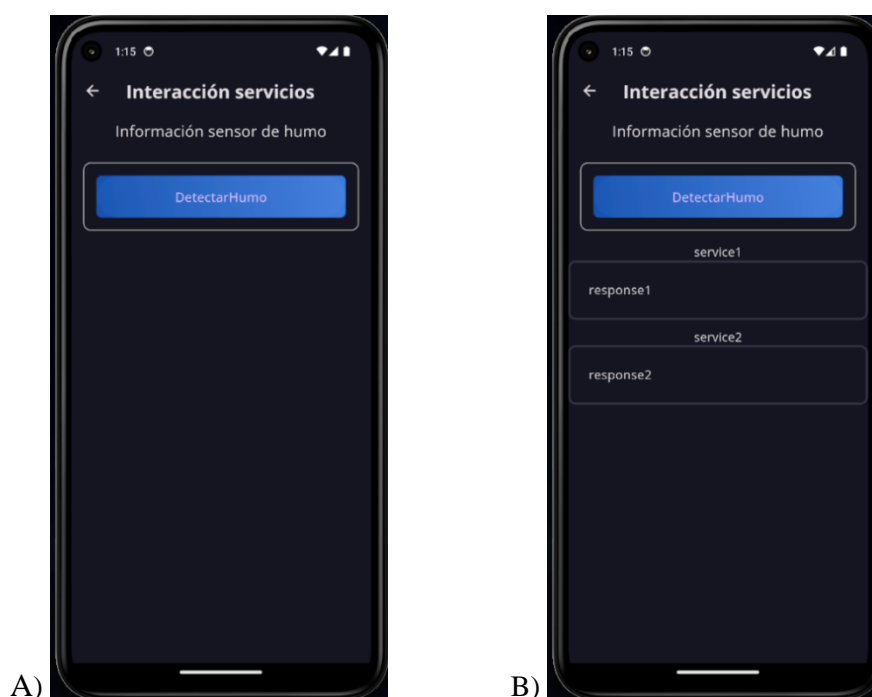
Funcionalidad: La página de servicios descubiertos en la figura 6, muestra categorías desplegadas basadas en el tipo de servicio establecido en la entidad TxtProperties de la información publicada por el servicio mDNS. Los tipos de servicio pueden ser: sensor, actuador o aplicación. Cada categoría incluye el número de dispositivos que ofrecen ese tipo de servicio. Al seleccionar una categoría, se despliega, la lista de servicios correspondientes. En cada tarjeta de servicio, se muestra la dirección IP del dispositivo IoT, el nombre del servicio publicado y el puerto donde está disponible el servicio anunciado.

Permite al usuario obtener una visión organizada y clara de los servicios disponibles en su entorno, facilitando la interacción con los dispositivos IoT según sus necesidades y perfil. Al categorizar los servicios, se mejora la usabilidad de la aplicación y se personaliza la experiencia del usuario al interactuar con dispositivos y servicios en la red local.

#### 4.3.1.4 Pagina servicio Mdns

**Figura 7**

*Pantalla interacción de servicios*



La Figura 7 ilustra la interfaz de la aplicación móvil cuando un usuario selecciona un servicio REST específico de la lista de servicios mDNS disponibles. En la figura, se observa un botón que representa un método GET del servicio REST asociado al servicio mDNS seleccionado. El título de la página se obtiene del atributo pageName definido en las propiedades TXT del servicio mDNS, proporcionando una descripción clara del contexto del servicio. Los botones que aparecen en la lista de servicios derivan del atributo apiService de cada servicio mDNS, reflejando así los diferentes endpoints REST disponibles para interacción.

Al presionar uno de estos botones, la aplicación utiliza la información escaneada previamente por el teléfono, específicamente los parámetros address, port y apiService, para realizar una petición GET al endpoint correspondiente gestionado por el middleware. La respuesta de la petición GET, compuesta por pares clave-valor definidos por el servicio REST, se presenta en la interfaz mediante etiquetas y campos de texto. Cada par clave-valor se asigna a una etiqueta descriptiva y a un campo de texto correspondiente, permitiendo al usuario visualizar de manera clara y estructurada los datos obtenidos del servicio REST.

## 4.3.2 Capa de Dominio

### 4.3.2.1 Entidad Mdns Service

**Figura 8**

*Fragmento entidad MdnsService.dart*

```
class MdnsService {  
  final String address;  
  final String serviceType;  
  final int port;  
  final String? mdnsName;  
  final TxtProperties txtProperties;  
}
```

La entidad de la figura 8 define la estructura para serialización y gestión de servicios mDNS. Se tienen las siguientes propiedades:

- address: Dirección IP del servicio mDNS publicado en red local.
- serviceType: Tipo de servicio mDNS. Como la interacción estrictamente se realiza con servicios RESTful, siempre se espera el tipo `_http._tcp.local`.
- port: Puerto en el que está publicado el servicio RESTful.
- mdnsName: Nombre del servicio, que puede actuar como dominio local.
- txtProperties: Objeto TxtProperties con información adicional.

### 4.3.2.2 Entidad TxtProperties

**Figura 9**

*Fragmento entidad TxtProperties.dart*

```
class TxtProperties {  
  final String location;  
  final String type;  
  
  TxtProperties({required this.location, required this.type});  
}
```

Define las propiedades de contexto obtenidas por la ubicación y el tiempo en que se realiza la lectura de servicios mDNS. Se detalla las propiedades presentes en la figura 9:

- location: Sigue un formato específico definido por el middleware, que se utiliza para estandarizar la forma en que se representa la ubicación en todo el sistema. El formato establecido es "CAMPUS: (Nombre Específico) /Edificio: (Nombre Específico) /Piso: (Nombre Específico) /Aula: (Nombre Específico)"
  - CAMPUS: Nombre del campus o ubicación general.
  - Edificio: Identificación del edificio dentro del campus.

- Piso: Nivel o planta del edificio.
- Aula: Número o identificación del aula o habitación específica.
- type: Indica el tipo de dispositivo IoT o servicio ofrecido.

Valores Posibles:

- "sensor": Si el dispositivo es un sensor que recopila datos del entorno (e.g., temperatura, humedad).
- "actuador": Si el dispositivo puede actuar o cambiar el estado del entorno (e.g., interruptores, válvulas).
- "aplicacion": Si se trata de una aplicación o servicio que no corresponde directamente a hardware físico.

### 4.3.2.3 Entidad Mdns Service Context

**Figura 10**

*Fragmento entidad MdnsServiceContext.dart*



```

class MdnsServiceContext {
  final String location;
  String timestamp;

  MdnsServiceContext({
    required this.location,
    required this.timestamp,
  });
}

```

Define la estructura para serializar las propiedades adicionales obtenidas al leer la publicación de servicios mDNS (ver figura 10).

- location: Representa el nombre del dispositivo Bluetooth Low Energy (BLE) escaneado. A diferencia de locacion de la entidad TxtProperties, el nombre del valor corresponde a la localizacion con formato del middleware aplicado hash sha256 y se toman los primeros 16 digitos.
- timestamp: Representa la fecha y hora exactas en que se realizó la lectura o escaneo del dispositivo BLE. Crucial para asegurar que la información de ubicación es actual y para sincronizar los servicios y datos contextuales en función del momento en que se obtuvo la ubicación. Esto es especialmente importante en entornos dinámicos donde la posición del usuario puede cambiar frecuentemente. Formato del Valor: Se almacena generalmente en formato de marca de tiempo estándar (e.g., YYYY-MM-DD HH:MM:SS)

### 4.3.3 Capa de Datos

#### 4.3.3.1 Repositorio Mdns Service Repository

Figura 11

Fragmento entidad *MDnsServiceRepositoryImpl.dart*

```
class MDnsServiceRepositoryImpl implements MDnsServiceRepository {
  final ConnectionChecker connectionChecker;
  final MDnsServiceNetworkDataSource mdnsServiceNetworkDataSource;
  final MDnsServiceLocalDataSource mdnsServiceLocalDataSource;
  final MDnsServiceRemoteDataSource mdnsServiceRemoteApi;
  final MDnsServiceContextDataSource mdnsServiceContextDataSource;
  final BeaconScanner beaconScanner;

  @override
  Future<Either<Failure, List<MDnsService>>> getMdnsServices() async {
    try {
      // Descubre servicios mDNS en la red local
      final mdnsServices = await
mdnsServiceNetworkDataSource.getMdnsServices();

      // Actualiza la base de datos local con los servicios
      descubiertos
      await _updateLocalDatabase(mdnsServices);

      // Verifica la conexión a Internet
      if (!await connectionChecker.isConnected) {
        return left(const NoInternetConnectionFailure());
      }

      // Envía servicios no sincronizados al servidor
      final unsyncedServices = await
mdnsServiceLocalDataSource.getUnsyncedMdnsServices();
      final serverResponse = await
mdnsServiceRemoteApi.sendMdnsServices(
        unsyncedServices, await _lastContextInfo());

      // Marca los servicios como sincronizados
      await
mdnsServiceLocalDataSource.markServicesAsSynced(unsyncedServices);

      // Retorna la lista de servicios desde el servidor
      return right(serverResponse);
    } on ServerException catch (e) {
      return left(ServerFailure(e.message, e.statusCode));
    } on MdnsScannerFailure catch (e) {
      return left(MdnsScannerFailure(e.errorMessage));
    }
  }

  Future<void> _updateLocalDatabase(List<MDnsServiceModel>
mdnsServices) async {
    // Implementación para actualizar la base de datos local
  }

  Future<MDnsServiceContextModel> _lastContextInfo() async {
    return await mdnsServiceContextDataSource.getLastInfo();
  }
}
```

Se centra en la interacción con los servicios mDNS descubiertos en la red local, la gestión de la información contextual y la comunicación con el middleware. A continuación, se detalla la implementación del repositorio MDnsServiceRepositoryImpl presente en la figura 11.

- ConnectionChecker: Comprueba el estado de la conectividad a Internet. Es esencial para determinar si es posible sincronizar los datos con el servidor remoto (middleware).
- MDnsServiceNetworkDataSource : Fuente de datos encargada de descubrir los servicios mDNS disponibles en la red local. Utiliza la librería multicast\_dns para realizar el escaneo.

- **MdnsServiceLocalDataSource** : Maneja el almacenamiento local de los servicios mDNS descubiertos. Utiliza una base de datos SQLite a través de la librería sqflite para persistir los datos.
  - Insertar nuevos servicios en la base de datos local.
  - Obtener servicios no sincronizados.
  - Marcar servicios como sincronizados después de enviarlos al servidor.
- **MdnsServiceRemoteApi** : Interactúa con el middleware para enviar y recibir servicios. Utiliza la librería dio para manejar las solicitudes HTTP.
  - Enviar los servicios mDNS descubiertos al middleware junto con la información contextual.
  - Recibir la lista de servicios autorizados y personalizados según el contexto del usuario.
- **MdnsServiceContextDataSource** : Gestiona la información de contexto de los servicios, como la ubicación y el timestamp. Almacena y recupera esta información para ser utilizada en la sincronización con el middleware.
- **BeaconScanner** : Escanea dispositivos Bluetooth Low Energy (BLE) cercanos para obtener la ubicación contextual del usuario, obtiene el dispositivo filtrado por su UUID más cercano en base a la señal RSSI. Utiliza la librería flutter\_blue\_plus.
- **GetMdnsServices** : punto central donde se coordina la obtención y sincronización de los servicios mDNS. A continuación, se detalla paso a paso su funcionalidad:
  1. Descubrir Servicios mDNS en la Red Local: Utiliza mdnsServiceNetworkDataSource para escanear la red local en busca de servicios mDNS.
  2. Actualizar la Base de Datos Local: Llama al método privado \_updateLocalDatabase() para insertar los nuevos servicios en la base de datos local y eliminar aquellos que ya no están disponibles.
  3. Verificar la Conexión a Internet: Utiliza connectionChecker para determinar si hay conexión a Internet antes de intentar sincronizar con el servidor.
  4. Enviar Servicios No Sincronizados al Servidor (Middleware): Obtiene los servicios que aún no han sido sincronizados utilizando mdnsServiceLocalDataSource.
- Obtiene la última información de contexto con \_lastContextInfo().
- Envía los servicios y el contexto al middleware a través de mdnsServiceRemoteApi.
  5. Marcar los Servicios como Sincronizados: Una vez que el servidor confirma la recepción, actualiza el estado de los servicios en la base de datos local para indicar que han sido sincronizados.
  6. Retornar la Lista de Servicios desde el Servidor: Retorna la lista de servicios autorizados y personalizados que el servidor ha devuelto.
  7. Manejo de Excepciones: Captura excepciones específicas como ServerException y MdnsScannerFailure, retornando un Failure apropiado.

La implementación del MDnsServiceRepositoryImpl es fundamental para:

- Integración de Servicios Locales y Remotos: Combina la información obtenida localmente con la proporcionada por el middleware, asegurando que el usuario acceda a servicios relevantes y autorizados.
- Adaptación al Contexto: Al incorporar información contextual (ubicación y timestamp), el sistema puede adaptarse dinámicamente a las necesidades y situaciones del usuario.
- Eficiencia y Rendimiento: Almacenar servicios localmente reduce la dependencia de la conectividad constante a Internet. Sincronizar solo los servicios no sincronizados optimiza el uso de recursos y ancho de banda.
- Mantenibilidad y Escalabilidad: La separación clara de responsabilidades y el uso de interfaces permiten que la implementación pueda ser modificada o ampliada sin afectar otras partes del sistema.

## 5. Middleware de automodelado de arquitecturas de IoT

El middleware constituye el componente central encargado de gestionar y procesar la información del sistema en tiempo real, integrando datos provenientes de dispositivos IoT y aplicaciones cliente. Desarrollado en Node.js y respaldado por PostgreSQL, su principal función es

comparar los modelos predefinidos con la información actual del sistema, permitiendo una adaptación dinámica según las condiciones operativas. Para facilitar su despliegue, el middleware cuenta con un archivo de configuración que permite ajustar parámetros de operación, incluyendo la integración de certificados de seguridad para habilitar la comunicación HTTPS; en ausencia de estos certificados, el sistema opera en modo HTTP. Este archivo de configuración también permite configurar la conexión a la base de datos sin exponer las credenciales directamente en el código, definir la dirección y el puerto de despliegue, así como establecer rutas para elementos visuales como el logo de la aplicación web y el título mostrado en el navegador. Además, el archivo de configuración es escalable, permitiendo agregar nuevas configuraciones como credenciales para servidores de correo u otros servicios según sea necesario.

El middleware desempeña tres funciones clave: primero, comparar el modelo en tiempo de diseño para verificar si el estado del sistema cumple con las reglas definidas en el modelo de diseño; segundo, generar modelos en tiempo de ejecución, adaptando el modelo almacenado en PostgreSQL en respuesta a cambios en el entorno; y tercero, implementar consultas y gestionar la conexión a la base de datos, ejecutando consultas y almacenando información para asegurar la consistencia y eficiencia en la toma de decisiones adaptativas. Además, el middleware utiliza sockets para mantener la sincronización de la información entre los diferentes componentes del sistema. Cada vez que recibe una petición de la aplicación móvil, notifica a la aplicación web mediante un socket cliente, permitiendo que la información se actualice de manera inmediata y manteniendo a los usuarios visualizando datos en tiempo real sin necesidad de recargar manualmente la página.

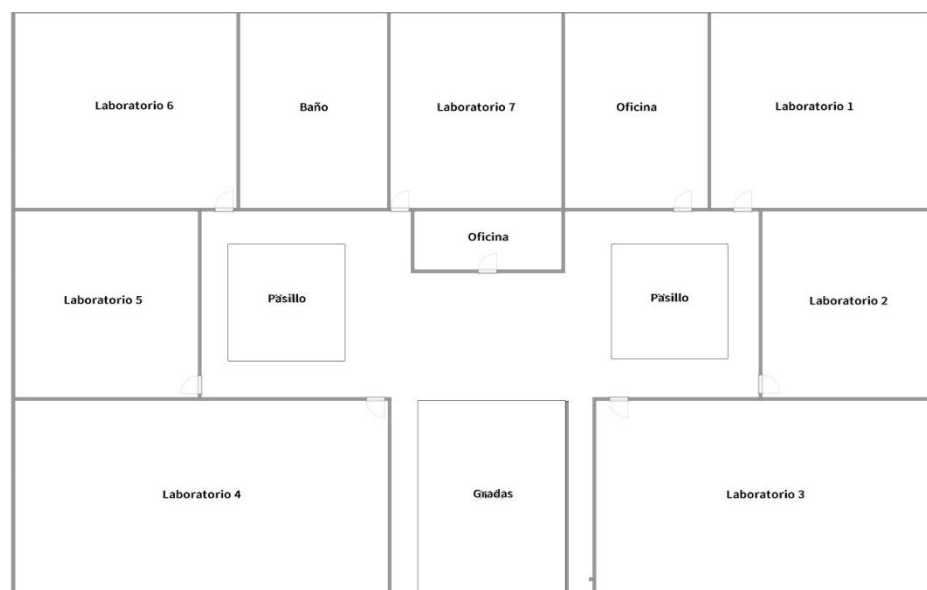
## 6. Estudio de caso

### 6.1 Escenario ilustrativo de IoT

Para evaluar y validar la funcionalidad de la aplicación desarrollada, se propone un estudio de caso en un contexto universitario real, específicamente en la Universidad del Azuay. El escenario seleccionado incluye el Campus Central, dentro del Edificio de Filosofía, en el Piso 4, donde se dispone de 7 laboratorios. Este entorno representado en la figura 12 ofrece un ambiente controlado y representativo para probar las capacidades de descubrimiento y gestión de servicios IoT mediante la aplicación móvil y el middleware desarrollado.

**Figura 12**

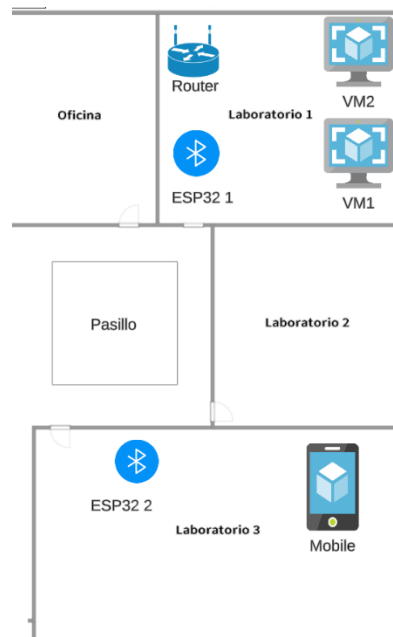
*Plano piso 4 de la facultad de filosofía*



La distribución de dispositivos se representa en el Diagrama de Red de Planos de Planta correspondiente a la figura 13. El espacio físico abarca el Laboratorio 1, el pasillo y el Laboratorio 3. En el Laboratorio 1 se encuentra un ESP32 que actúa como dispositivo Bluetooth, facilitando la

comunicación inalámbrica con la aplicación móvil. Por otro lado, en el Laboratorio 3 se dispone de un ordenador que funciona como router inalámbrico, gestionando una red WiFi privada con DHCP para la asignación dinámica de direcciones IP. Además, en este laboratorio hay un computador que alberga dos máquinas virtuales (VM1 y VM2) y un ESP32 con servicio Bluetooth, proporcionando servicios adicionales en la red local. Finalmente, el usuario utiliza un móvil con la aplicación de descubrimiento para interactuar y gestionar los servicios IoT disponibles en el entorno. Los equipos involucrados se describen en la tabla 1.

**Figura 13**  
*Distribución de dispositivos*





**Tabla 1**  
*Lista de equipamientos empleados en el caso de estudio*

Equipo	Función	Descripción
ESP32	Hardware: Dispositivo BLE	Ambos dispositivos ESP32 operan en la banda ISM de 2,4 GHz y emiten señales BLE con un intervalo de publicación de 1ms, lo que permite una detección rápida y precisa por parte de la aplicación móvil.
	Software: Anuncio de Ubicación	Cada ESP32 anuncia su ubicación específica mediante el nombre del servicio BLE como un texto de 16 caracteres codificado con sha256, siguiendo el formato establecido por el middleware para facilitar la categorización y gestión contextual de los servicios IoT.
Máquina virtual (VM1, VM2)	Hardware: Dispositivo Iot	Simulan dispositivos IoT similares a un Raspberry Pi, en lugar de usar el sistema operativo Linux específico de Raspberry Pi Raspbian, emplean Ubuntu 20.2
	Software: Avahi	Para la publicación de servicio mDNS en la red se emplea Avahi, el cual es un sistema que facilita el descubrimiento de servicios en una red local mediante el protocolo mDNS/DNS-SD. Esto permite conectar una laptop o computadora a una red y descubrir otros dispositivos (Poettering & Llod, 2023)
	Software: FastAPI	Para construir APIs web en Python se emplea el framework FastAPI, que es un framework moderno y de alto rendimiento para construir APIs con Python basado en anotaciones de tipos estándar (Ramírez, s.f)
	Software: Uvicorn	Para la ejecución de la aplicación web con los servicios REST de FastAPI, se emplea Uvicorn, una implementación de servidor web ASGI para Python. Uvicorn soporta HTTP/1.1 y WebSockets, facilitando la integración con frameworks asíncronos. (Trylesinski, s.f)
	Software: Simulación de Sensores IoT	Los servicios REST publicados simulan la interacción con sensores, permitiendo probar la capacidad de la aplicación para interactuar con servicios IoT mediante peticiones HTTP.
Router (PC con Windows 10)	Hardware: Red WiFi Privada	El router crea y gestiona una red WiFi privada con DHCP, asignando direcciones IP dinámicamente a los dispositivos conectados. Proporciona una conexión estable y segura para la transmisión de datos entre la aplicación móvil, los dispositivos BLE y los servicios mDNS publicados por las máquinas virtuales.
Teléfono inteligente	Hardware: Dispositivo Móvil	Dispositivo móvil Samsung Galaxy A32 con Android 13, soporta Wi-Fi 802.11 a/b/g/n/ac, dual-band, Wi-Fi Direct y Hotspot Bluetooth 5.0.
	Software: Aplicación de Contexto	Ejecuta la aplicación de descubrimiento desarrollada para gestionar servicios IoT mediante mDNS y BLE. La aplicación permite al usuario iniciar el descubrimiento de servicios, visualizar los servicios disponibles a través de una interfaz de usuario intuitiva, y obtener detalles específicos de cada servicio IoT. Además, facilita la interacción y gestión en tiempo real de los servicios descubiertos, adaptándose al contexto.


## 6.2 Instanciación y evaluación empírica de la solución tecnológica

### 6.2.1 Configuración dispositivo IoT

La simulación de los dispositivos IoT, se emplean mediante máquinas virtuales con Ubuntu, en la cuales se configuran los servicios mDNS utilizando Avahi Services. El servicio publicado en la VM1 representa un sensor de temperatura, mientras que el servicio en la VM2 simula un sensor de humo, ambos siguiendo los estándares y propiedades definidos en la aplicación móvil y el middleware. El archivo de configuración del servicio mDNS, denominado `iot1.service`, está estructurado de la siguiente manera:

**Figura 14**

*Fragmento archivo de configuración publicación de servicio mdns.*



```
iot1.service:
<?xml version="1.0" standalone='no'?>
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
  <name replace-wildcards="yes">%h</name>
  <service>
    <type>_http._tcp</type>
    <port>8000</port>
    <txt-record>location=CAMPUS:Campus Universidad del Azuay/
Edificio: Administracion/Piso:3/Aula:306</txt-record>
    <txt-record>type=sensor</txt-record>
  </service>
</service-group>
```

La configuración presente en la figura 14 publica un servicio mDNS de tipo HTTP en el puerto 8000, con registros TXT que especifican la ubicación y el tipo de servicio. La ubicación sigue el formato estandarizado por el middleware "CAMPUS: Campus Universidad del Azuay/Edificio: Administración/Piso:3/Aula:306" y el tipo de servicio se clasifica como sensor, permitiendo a la aplicación móvil categorizar y gestionar adecuadamente los dispositivos descubiertos.

El archivo de configuración se encuentra ubicado en el directorio `/etc/avahi/services/`, que define el tipo de servicio, el puerto y los registros TXT correspondientes. El contenido del archivo sigue la estructura estandarizada por Avahi para anunciar servicios en la red local. Una vez creado el archivo, se ejecutan los comandos de la figura 15 para copiarlo al directorio estándar y reiniciar el daemon de Avahi, asegurando así la correcta publicación del servicio

**Figura 15**

*Comandos de inicialización servicio Avahi.*



```
sudo cp iot1.service /etc/avahi/services/
sudo systemctl restart avahi-daemon
```

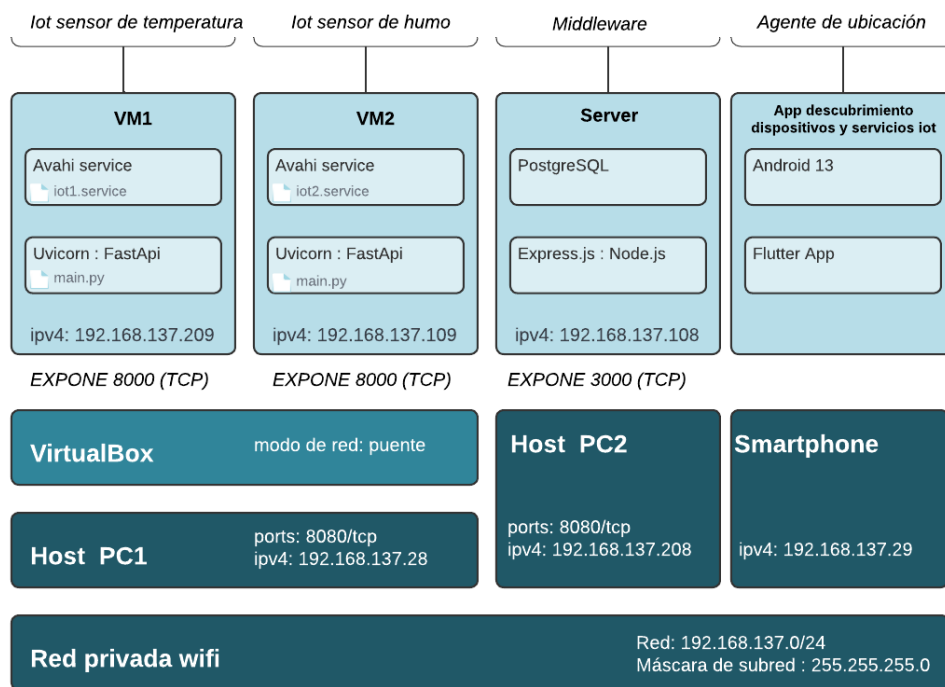
Se publican los servicios REST definidos en el archivo `main.py`, como se muestra en la Figura 14, donde se presenta un fragmento de dichos servicios a publicar. En la ruta GET, se especifica la ruta establecida por el middleware según el tipo de servicio, ya sea temperatura o humo. En ambos casos, se crean rutas que devuelven valores simulados de retorno correspondientes a grados centígrados para temperatura y microgramos por metro cúbico para humo. Para ejecutar el servidor web con la aplicación de FastAPI, se utiliza el comando `: uvicorn main:app --port 8000`. Este comando inicia el servidor web ASGI proporcionado por Uvicorn, permitiendo que la aplicación FastAPI escuche en el puerto 8000 y gestione las solicitudes HTTP entrantes.

Tras la inicialización de Avahi y la ejecución del servidor web Uvicorn, es importante señalar que las dos máquinas virtuales se ejecutan en VirtualBox con la configuración de red en puente. Esto implica que las direcciones IP asignadas a las máquinas virtuales no dependen de la máquina host, sino

que son gestionadas por el DHCP de la red WiFi compartida del laboratorio. En consecuencia, cada máquina virtual recibe una dirección IP única proporcionada por el router inalámbrico del laboratorio, asegurando una comunicación fluida y sin conflictos en la red local. Además, el dispositivo móvil utilizado para la aplicación de contexto obtiene su dirección IP también a través de este servidor DHCP, al igual que el servidor que alberga el middleware. En la Figura 16, se presenta un esquema detallado de cómo se distribuyen los distintos componentes en la red, incluyendo las direcciones IP de las dos máquinas virtuales, el teléfono móvil y el servidor del middleware.

**Figura 16**

*Diagrama de componentes en la red perteneciente al estudio de caso.*



## 6.2.2 Configuración dispositivo Bluetooth

Para anunciar la ubicación de manera física, se utiliza un dispositivo ESP32 configurado como beacon Bluetooth Low Energy (BLE). La configuración del ESP32 se realiza mediante el siguiente código presente en la figura 17.

**Figura 17**

*Fragmento archivo de configuración esp32 como dispositivo bluetooth*

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <BLEBeacon.h>

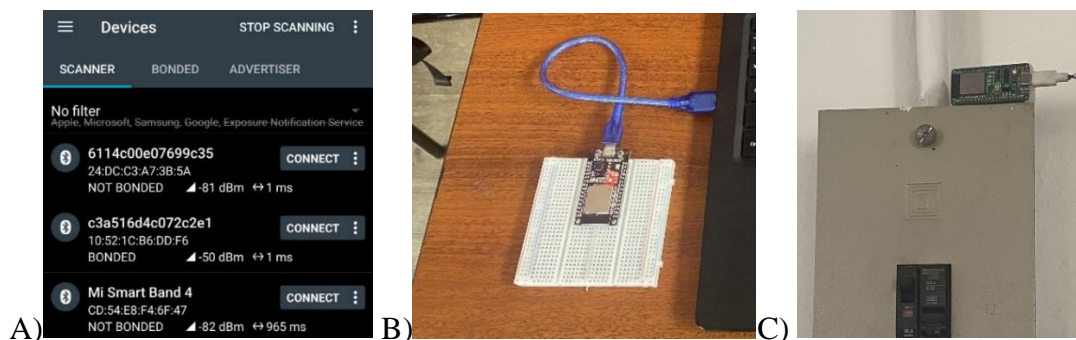
#define DEVICE_NAME           "86f451191dc35da4e"
#define SERVICE_UUID          "7A0247E7-8E88-409B-A959-AB5092DDB03E"
#define BEACON_UUID           "2D7A9F0C-E0E8-4CC9-A71B-A21DB2D034A1"
#define BEACON_UUID_REV      "A134D0B2-1DA2-1BA7-C94C-E8E00C9F7A2D"
#define CHARACTERISTIC_UUID   "82258BAA-DF72-47E8-99BC-B73D7ECD08A5"
```

Este dispositivo BLE emite señales que pueden ser detectadas por la aplicación móvil mediante la librería flutter\_blue\_plus, permitiendo así la obtención de la ubicación física en tiempo real dentro del aula. La configuración del ESP32 asegura que cada señal emitida contenga identificadores únicos y pertinentes que facilitan la correlación entre la ubicación física y los servicios mDNS descubiertos. Además, en la aplicación se filtra únicamente los dispositivos BLE cuyo UUID sea "7A0247E7-8E88-409B-A959-AB5092DDB03E". La configuración del ESP32 se codifica en un archivo de tipo ino, el cual es compilado y cargado al dispositivo, mediante el programa Arduino IDE.

Para verificar la correcta configuración y funcionamiento de los dispositivos BLE, se utiliza la aplicación nrfSCAN, la cual permite observar y analizar las propiedades de los dispositivos BLE (ESP32) en tiempo real. Esto garantiza que los servicios anunciados y las señales BLE emitidas cumplan con los estándares y especificaciones de UID requeridas. En la Figura 18a se muestra la lectura del ESP32 como dispositivo BLE, donde el nombre "6114c00e07699c35" corresponde a la codificación SHA-256 de la localización "CAMPUS: Campus Universidad del Azuay/Edificio: Filosofía/Piso: 4/Aula: Laboratorio 3", publicada en el ESP32 ubicado en el Laboratorio 3 (Figura 18b). Por otro lado, el nombre "c3a516d4c072c2e1" representa la codificación SHA-256 de la localización "CAMPUS: Campus Universidad del Azuay/Edificio: Filosofía/Piso: 4/Aula: Laboratorio 1", publicada en el ESP32 situado en el Laboratorio 1 (Figura 18c).

**Figura 18**

*Verificación e instanciación operativa de los dispositivos esp32 como dispositivos bluetooth*



### 6.2.3 Parámetros de Evaluación Empírica

Para evaluar empíricamente la solución tecnológica desarrollada, que incluye la aplicación móvil y el middleware, se han establecido una serie de parámetros de evaluación que se alinean directamente con los objetivos específicos de la tesis. La elección de estos criterios se fundamenta en la necesidad de

validar que el prototipo cumple con los requisitos planteados y demuestra su efectividad en un entorno realista. A continuación, se detallan los parámetros seleccionados en la tabla 2.

**Tabla 2**  
*Especificación de los parámetros de evaluación empírica*

<b>Parámetro</b>	<b>Descripción</b>	<b>Método de evaluación</b>
Precisión en la Localización	Evalúa la exactitud con la que la aplicación móvil determina la ubicación física del usuario dentro del aula utilizando señales BLE emitidas por los dispositivos ESP32.	Comparar las ubicaciones detectadas por la aplicación con posiciones reales verificadas manualmente.
Precisión en el Descubrimiento de Servicios IoT	Mide la capacidad de la aplicación móvil para identificar correctamente los servicios IoT disponibles en distintas ubicaciones (Laboratorio 1 y Laboratorio 3) y la ausencia de detección en áreas sin dispositivos BLE (pasillo)	Comparar los servicios descubiertos por la aplicación con un listado predefinido de dispositivos IoT presentes en cada ubicación.
Interacción Correcta con Servicios REST	Evalúa la capacidad de la aplicación móvil para interactuar correctamente con los servicios REST publicados por las máquinas virtuales, incluyendo la realización de peticiones GET y la correcta presentación de los datos simulados.	Realizar pruebas de interacción con los servicios REST ("Humo" y "Temperatura") y verificar que los valores retornados coincidan con los datos simulados.
Comportamiento temporal (iso 25010)	Evalúa cómo la aplicación móvil actualiza su interfaz en función de los resultados del escaneo BLE y la detección de servicios mDNS, incluyendo la presentación de mensajes según el estado esperado.	Observar y documentar la sincronización entre el estado del proceso de escaneo y la actualización de la interfaz, verificando la correcta presentación de mensajes y animaciones en cada etapa.
Estabilidad del Sistema en Diferentes Ubicaciones	Observa el comportamiento del sistema al utilizar la aplicación móvil en distintas ubicaciones (Laboratorio 1, pasillo, Laboratorio 3), verificando la consistencia en la detección y gestión de servicios.	Monitorear el rendimiento y la estabilidad del sistema durante el uso en cada ubicación, identificando posibles inconsistencias o fallos.

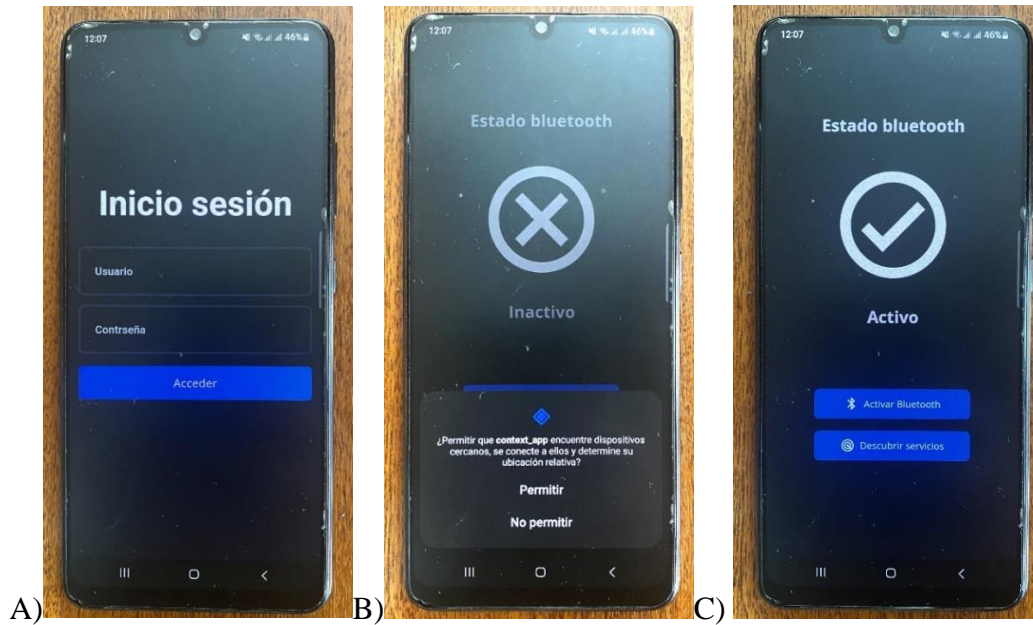
#### 6.2.4 Ejecución evaluación empírica

Tras la publicación de los servicios mDNS, la ejecución del servidor web Uvicorn con los servicios REST en cada máquina virtual y el anuncio de ubicación mediante Bluetooth Low Energy por parte de los dos dispositivos ESP32, se procede a utilizar la aplicación móvil de contexto. La evaluación empírica de la solución tecnológica desarrollada se llevó a cabo en tres ubicaciones específicas dentro de la Universidad del Azuay: Laboratorio 1, el Pasillo y Laboratorio 3. Este proceso inició en el Laboratorio 1, donde se realizó la configuración inicial y se verificaron las interacciones básicas de la aplicación móvil con los servicios IoT. Posteriormente, se procedió al Pasillo para evaluar el comportamiento de la aplicación en un entorno sin dispositivos BLE activos, y finalmente, se trasladó al Laboratorio 3 para validar la funcionalidad completa en un entorno con distinta locación y servicio mDNS asignado.

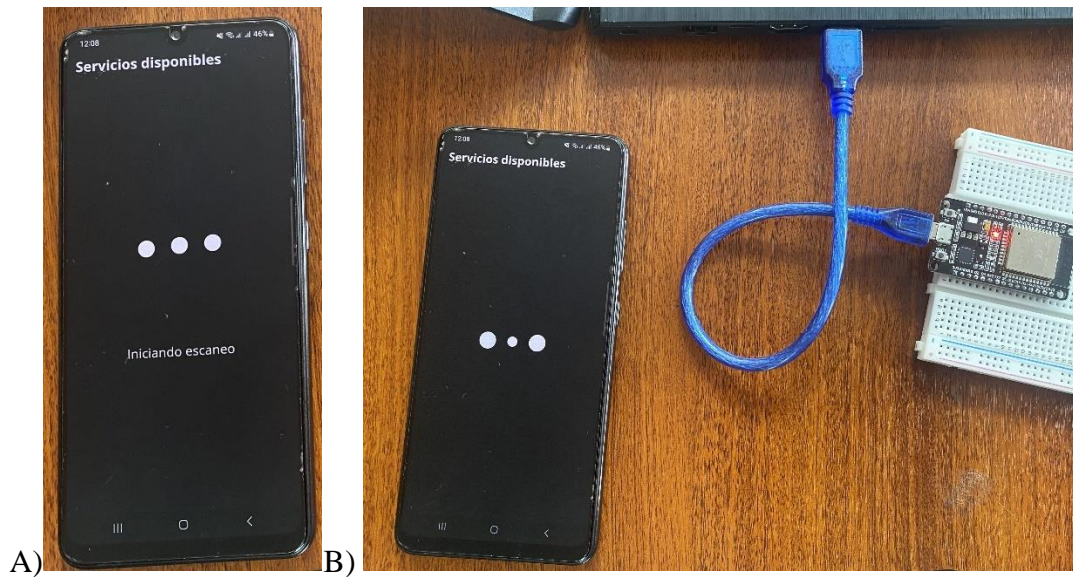
##### Laboratorio 1

Se ingresa a la aplicación móvil mediante las credenciales de inicio de sesión (Figura 19a). La aplicación solicita permisos de conexión para acceder a dispositivos cercanos y determinar la ubicación relativa del usuario (Figura 19b). Una vez otorgados los permisos, se activa el Bluetooth (Figura 19c). Durante el proceso de escaneo, se mostraron mensajes de estado como "Iniciando escaneo" (ver Figura 20a), seguido de una espera de 14 segundos (ver Figura 20b) para completar dos escaneos consecutivos, necesarios para confirmar la detección de los dispositivos BLE con ubicación, retribución de servicios mDNS publicados en la red, envío de la información al middleware y recepción de los servicios autorizados.

**Figura 19**  
*Vistas de acceso y permiso.*

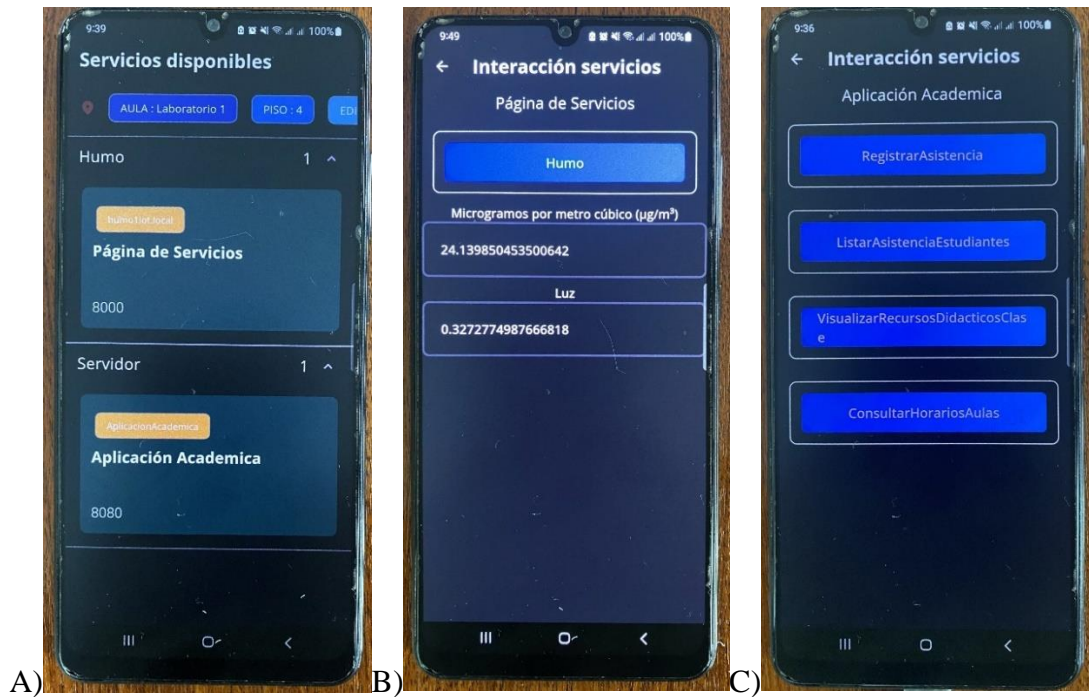


**Figura 20**  
*Vistas de carga*



Posteriormente, la aplicación despliega la ubicación actual proporcionada por el dispositivo BLE con el nombre "6114c00e07699c35", correspondiente a la ubicación "AULA: Laboratorio 1", "PISO: 4", "EDIFICIO: FILOSOFÍA", "CAMPUS: Campus Universidad del Azuay". En la sección desplegable se listan dos servicios: "HUMO" y "Servidor" (ver Figura 21a). Al seleccionar "HUMO", se accede a la página de interacción del servicio, donde se muestra un botón con el nombre del servicio "Humo". Al presionar el botón se obtiene los valores simulados de "Microgramos por metro cúbico ( $\mu\text{g}/\text{m}^3$ )" con un valor de 24.13 y "Luz" con un valor de 0.32 (ver Figura 21b). Asimismo, al seleccionar "Servidor", se presentan las rutas GET de la aplicación académica, permitiendo la interacción con funcionalidades como "RegistrarAsistencia", "ListarAsistenciaEstudiantes", "VisualizarRecursosDidácticosClase" y "Consultar Horarios Aulas" (ver Figura 21c).

**Figura 21**  
*Vistas resultado de servicios disponibles*



**Pasillo**

Se procede a utilizar la aplicación móvil para evaluar su comportamiento en una ubicación sin dispositivos BLE activos. Al iniciar el proceso de escaneo, la pantalla de la aplicación muestra el título "Servicios disponibles" acompañado de una animación de escaneo con cuatro puntos y el mensaje "Esperando un dispositivo Bluetooth cercano para determinar la ubicación". Tras 14 segundos de espera, la aplicación no detectó ningún dispositivo BLE, lo que resultó en la ausencia de servicios mDNS listados (ver Figura 22).

**Figura 22**  
*Vistas escaneo de dispositivos BLE*

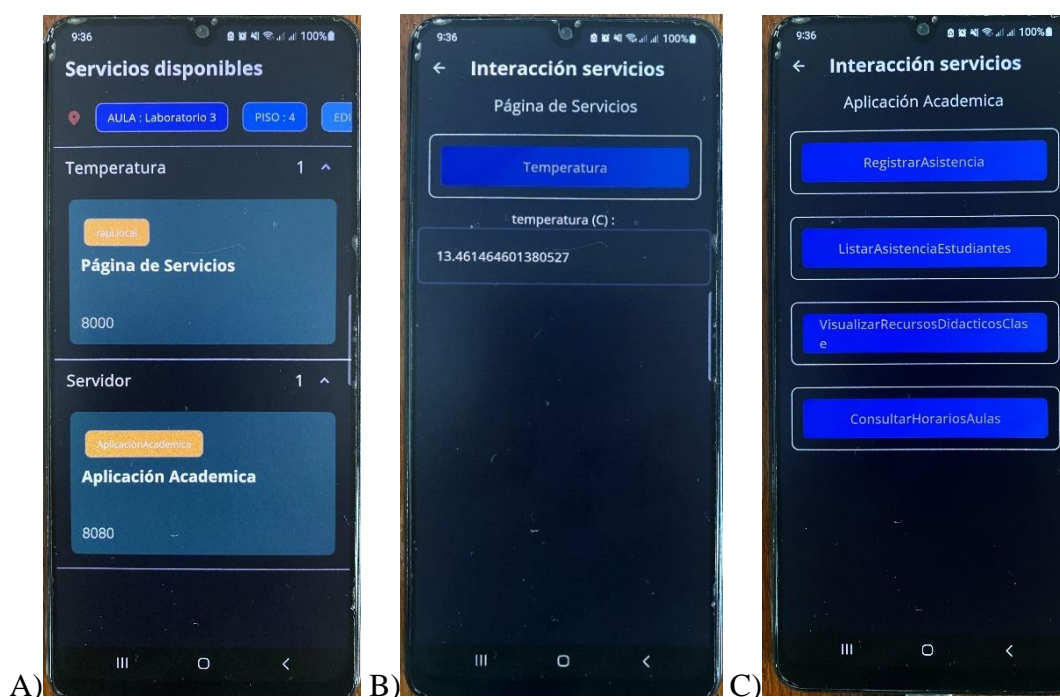


### Laboratorio 3

Finalmente, en el Laboratorio 3, La aplicación realizó el escaneo de dispositivos BLE, detectando nuevamente la ubicación relativa del usuario. Por lo que la pantalla automáticamente actualizó la vista, donde se mostraron tres puntos de carga seguidos de la lista de servicios autorizados por el middleware (ver Figura 23a). Los servicios detectados incluyeron "Temperatura" y "Servidor", correspondientes a los servicios publicados por la VM1 y el servidor del middleware, respectivamente. Al seleccionar "Temperatura", la aplicación navega a la página de interacción del servicio, mostrando el título "Página de Servicios" y el botón "Temperatura". Al presionar este botón, se realiza una petición GET al servicio de la VM1, retornando el valor de "Temperatura (°C)" con un valor de 13.4614 (ver Figura 23b). De igual manera, al seleccionar "Servidor", se presentaron las rutas GET de la aplicación académica, permitiendo la interacción con las funcionalidades previamente mencionadas (ver Figura 23c).

**Figura 23**

*Vistas escaneo de dispositivos BLE*



#### 6.2.5 Evaluación de Parámetros

A continuación, se detallan cada uno de estos parámetros, su descripción, método de evaluación y los resultados obtenidos en la tabla 3.



**Tabla 3***Resultado de evaluación empírica*

<b>Parámetro</b>	<b>Método de evaluación</b>	<b>Resultado</b>
Precisión en la Localización	Se compararon las ubicaciones detectadas por la aplicación con posiciones reales verificadas manualmente en cada ubicación. En el Laboratorio 1, la aplicación identificó correctamente la ubicación "AULA: Laboratorio 1", "PISO: 4", "EDIFICIO: FILOSOFÍA", "CAMPUS: Campus Universidad del Azuay" mediante el dispositivo BLE con el nombre "6114c00e07699c35". En el Laboratorio 3, se detectó igualmente la ubicación precisa mediante el dispositivo BLE correspondiente. En el Pasillo, donde no hay dispositivos BLE activos, la aplicación no detectó ninguna ubicación, lo cual es el comportamiento esperado.	La aplicación muestra precisión en la localización en las áreas con dispositivos BLE activos (Laboratorio 1 y 3) y correctamente no detectó ubicaciones en el pasillo, demostrando la funcionalidad operativa del sistema de localización basado en BLE.
Precisión en el Descubrimiento de Servicios IoT	Se compararon los servicios descubiertos por la aplicación con los dispositivos IoT (VM1 y VM2) presentes en cada ubicación. En el Laboratorio 1, la aplicación detectó correctamente los servicios "HUMO" y "Servidor", correspondientes a los servicios mDNS publicados por la VM2 y el middleware, respectivamente. En el Laboratorio 3, se detectaron los servicios "Temperatura" y "Servidor", asociados a la VM1 y el middleware. En el Pasillo, donde no existen dispositivos BLE activos, la aplicación no detectó ningún servicio IoT, confirmando la precisión en la detección basada en la presencia de dispositivos BLE.	La aplicación demuestra precisión en el descubrimiento de servicios IoT en las ubicaciones con dispositivos BLE activos y correctamente no identificó servicios en el pasillo, alineándose con los objetivos de detección precisa y contextual de servicios.
Interacción Correcta con Servicios REST	Se realizaron pruebas de interacción con los servicios REST "Humo" en el Laboratorio 1 y "Temperatura" en el Laboratorio 3, verificando que los valores retornados coincidan con los datos simulados. Al presionar el botón "Humo" en el Laboratorio 1, la aplicación obtuvo los valores de "Microgramos por metro cúbico (ug/m <sup>3</sup> )" con un valor de 24.13 y "Luz" con un valor de 0.32, correspondientes a los servicios REST de la VM2. En el Laboratorio 3, al seleccionar "Temperatura", se obtuvo el valor de "Temperatura (°C)" con 13.4614, alineado con los datos simulados de la VM1.	Las interacciones con los servicios REST fueron correctas, retornando los valores simulados esperados, lo que confirma la correcta implementación y funcionamiento de las APIs RESTful mediante FastAPI y Uvicorn.
Eficiencia en la Actualización de la Interfaz de Usuario	Se observó y documentó la sincronización entre el estado del proceso de escaneo y la actualización de la interfaz en cada ubicación. En el Laboratorio 1 y Laboratorio 3, la interfaz mostró correctamente los mensajes de "Iniciando escaneo" y "Esperando un dispositivo Bluetooth cercano para determinar la ubicación", seguidos por la presentación de los servicios descubiertos después de 14 segundos. En el Pasillo, la interfaz mostró adecuadamente la animación de escaneo y el mensaje de espera sin detectar dispositivos BLE, manteniendo la consistencia en la actualización de la interfaz.	La interfaz de usuario se actualizó de manera eficiente y precisa en todas las etapas del proceso de escaneo y descubrimiento, proporcionando coherencia con los estados del contexto.
Estabilidad del Sistema en Diferentes Ubicaciones	Se monitoreó el rendimiento y la estabilidad del sistema durante el uso en cada ubicación, identificando posibles inconsistencias o fallos. En todas las ubicaciones evaluadas, la aplicación funcionó de manera estable, detectando y gestionando correctamente los servicios IoT cuando estaba presente la señal BLE y no presentando fallos en el pasillo donde no había dispositivos BLE activos.	El sistema demuestra estabilidad y consistencia en diferentes entornos físicos, adaptándose a la presencia o ausencia de dispositivos BLE sin presentar errores o comportamientos anómalos.

## 7. Conclusiones

Los resultados de la evaluación empírica demuestran el cumplimiento del objetivo general del presente estudio, que es desarrollar un prototipo de una aplicación móvil basada en modelos en tiempo de ejecución para el descubrimiento de entidades IoT físicas y digitales. Gracias a la información recopilada en el marco teórico y en el estudio de caso, se pudo estructurar y construir este prototipo de aplicación móvil. La integración de tecnologías como FastAPI y Uvicorn para la implementación de servicios RESTful, junto con el uso de Avahi Services para la publicación de servicios mDNS, permitió una interacción fluida y precisa con los dispositivos IoT simulados en el entorno universitario. Además, la capacidad de la aplicación para detectar y gestionar servicios IoT en diferentes ubicaciones, como los Laboratorios 1 y 3 y el Pasillo, validó la efectividad de la solución tecnológica propuesta en condiciones reales de uso.

El uso de una arquitectura orientada a eventos resultó ser la herramienta adecuada para acoplar correctamente los eventos de descubrimiento de entidades BLE e IoT. Esta arquitectura permitió reflejar estados coherentes en la aplicación y facilitar la integración con el middleware, el cual gestiona los modelos en tiempo de ejecución, así como los servicios disponibles y sus características. La arquitectura orientada a eventos aseguró una comunicación eficiente entre los componentes del sistema, permitiendo una gestión dinámica y adaptable de los servicios IoT descubiertos. Esto no solo mejoró la funcionalidad y usabilidad de la aplicación móvil, sino que también garantizó una interacción robusta con los servicios mDNS y RESTful publicados por las máquinas virtuales. En resumen, la solución tecnológica desarrollada no solo cumple con los objetivos planteados, sino que también demuestra ser una herramienta para la gestión y descubrimiento de servicios IoT en entornos reales como el de la Universidad del Azuay. Los resultados obtenidos respaldan la operatividad de la arquitectura y las tecnologías empleadas, ofreciendo un modelo para futuras mejoras y adaptaciones en contextos similares.

## 8. Recomendaciones

A pesar de las optimizaciones implementadas, como el almacenamiento de información de servicios mDNS escaneados en el teléfono para evitar peticiones constantes al middleware, se recomienda modificar la librería utilizada para escanear servicios mDNS. Adaptar la librería específicamente a este caso de uso permitiría eliminar propiedades innecesarias que consumen memoria sin ser accedidas, optimizando así el rendimiento. Además, sería beneficioso realizar una comparativa del uso de la red en modo broadcast para el descubrimiento de servicios, explorando alternativas que puedan mejorar la eficiencia y reducir el consumo de ancho de banda, lo que contribuiría a una gestión más eficiente de los recursos de red.

En cuanto a la interacción con los servicios RESTful, se sugiere ampliar la funcionalidad para incluir métodos tipo POST, incorporando campos de ingreso en la interfaz de usuario y ajustando el middleware para gestionar estos nuevos parámetros. Esto permitiría una interacción más dinámica y completa con los servicios IoT. Asimismo, es fundamental implementar medidas de seguridad robustas, como la encriptación de datos y la autenticación, para proteger la comunicación entre la aplicación móvil, el middleware y los dispositivos IoT. Estas mejoras no solo aumentarían la seguridad del sistema, sino que también asegurarían la confidencialidad e integridad de la información manejada, alineándose con las mejores prácticas en entornos IoT.

## 9. Referencias

- Alfonso, I., Garcés, K., Castro, H., & Cabot, J. (2023). A model-based infrastructure for the specification and runtime execution of self-adaptive IoT architectures [Una infraestructura basada en modelos para la especificación y ejecución en tiempo de ejecución de arquitecturas de IoT autoadaptativas]. *Computing*, 105(9), 1883-1906. <https://doi.org/10.1007/s00607-022-01145-7>
- Bencomo, N., Götz, S., & Song, H. (2019). Models@run.time: A guided tour of the state of the art and research challenges [Models@run.time: una visita guiada al estado del arte y los desafíos de la investigación]. *Software & Systems Modeling*, 18(5), 3049-3082. <https://doi.org/10.1007/s10270-018-00712-x>
- Bézivin, J. (2004). In Search of a Basic Principle for Model Driven Engineering [En busca de un principio básico para la ingeniería basada en modelos]. *The European Journal for the Informatics Professional*, V(2), 21-25. NOVÁTICA.

- Blair, G., Bencomo, N., & France, R. B. (2009). Runtime adaptation mechanisms that leverage software models extend the applicability of model-driven engineering techniques to the runtime environment. [Los mecanismos de adaptación del tiempo de ejecución que aprovechan los modelos de software extienden la aplicabilidad de las técnicas de ingeniería basadas en modelos al entorno de ejecución.]. *Computer*, 42(10), 22-27. <https://doi.org/10.1109/MC.2009.326>
- Bonetto, R., Bui, N., Lakkundi, V., Olivereau, A., Serbanati, A., & Rossi, M. (2012). Secure communication for smart IoT objects: Protocol stacks, use cases and practical examples [Comunicación segura para objetos inteligentes de IoT: protocolos, casos de uso y ejemplos prácticos]. *2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 1-7. <https://doi.org/10.1109/WoWMoM.2012.6263790>
- Bouhamed, M. M., Díaz, G., Chaoui, A., Kamel, O., & Nouara, R. (2021). Models@Runtime: The Development and Re-Configuration Management of Python Applications Using Formal Methods [Models@Runtime: Gestión del desarrollo y la reconfiguración de aplicaciones Python mediante métodos formales]. *Applied Sciences*, 11(20), 9743. <https://doi.org/10.3390/app11209743>
- Chen, G., Zhang, Y., Chen, N.-S., & Fan, Z. (2016). Context-Aware Ubiquitous Learning in Science Museum with iBeacon Technology [Aprendizaje ubicuo y contextualizado en un museo de ciencias con tecnología iBeacon]. En M. J. Spector, B. B. Lockee, & M. D. Childress (Eds.), *Learning, Design, and Technology* (pp. 1-24). Springer International Publishing. [https://doi.org/10.1007/978-3-319-17727-4\\_5-1](https://doi.org/10.1007/978-3-319-17727-4_5-1)
- Cheshire, S., & Krochmal, M. (2013). *Multicast DNS* (Standards Track 2070-1721). Internet Engineering Task Force (IETF). <https://datatracker.ietf.org/doc/html/rfc6762>
- Fondement, F., & Silaghi, R. (2004). *Defining Model Driven Engineering Processes [Definición de procesos de ingeniería basados en modelos]*.
- France, R., & Rumpe, B. (2007). Model-driven Development of Complex Software: A Research Roadmap [Desarrollo de software complejo basado en modelos: una hoja de ruta para la investigación]. *Future of Software Engineering (FOSE '07)*, 37-54. <https://doi.org/10.1109/FOSE.2007.14>
- Gomez-Torres, E., & Lujan-Mora, S. (2017). An Approach of Context-Aware Mobile Applications for Internet of Things. *2017 International Conference on Information Systems and Computer Science (INCISCOS)*, 41-48. <https://doi.org/10.1109/INCISCOS.2017.65>
- Hendriks, S. (2016). *The Internet of Things [El internet de las cosas]* [Master Thesis, Utrecht University]. <https://studenttheses.uu.nl/handle/20.500.12932/23719>
- Hirsch, C., Davoli, L., Grosu, R., & Ferrari, G. (2023). DynGATT: A dynamic GATT-based data synchronization protocol for BLE networks [DynGATT: un protocolo dinámico de sincronización de datos basado en GATT para redes BLE]. *Computer Networks*, 222, 109560. <https://doi.org/10.1016/j.comnet.2023.109560>
- Hutchinson, J., Rouncefield, M., & Whittle, J. (2011). Model-driven engineering practices in industry [Prácticas de ingeniería basadas en modelos en la industria]. *Proceedings of the 33rd International Conference on Software Engineering*, 633-642. <https://doi.org/10.1145/1985793.1985882>
- Kaiser, D., & Waldvogel, M. (2014). Efficient Privacy Preserving Multicast DNS Service Discovery [Descubrimiento eficiente de servicios DNS de multidifusión que preservan la privacidad]. *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on CyberSpace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICSS)*, 1229-1236. <https://doi.org/10.1109/HPCC.2014.141>
- Khalil, A., & Dingel, J. (2018). Optimizing the Symbolic Execution of Evolving Rhapsody Statecharts [Optimización de la ejecución simbólica de diagramas de estados de Evolving Rhapsody]. En *Advances in Computers* (Vol. 108, pp. 145-281). Elsevier. <https://doi.org/10.1016/bs.adcom.2017.09.003>
- Khalil, K., Elgazzar, K., Seliem, M., & Bayoumi, M. (2020). Resource discovery techniques in the internet of things: A review. *Internet of Things*, 12, 100293. <https://doi.org/10.1016/j.iot.2020.100293>
- Li, S., Xu, L. D., & Zhao, S. (2015). The internet of things: A survey [El internet de las cosas: encuesta]. *Information Systems Frontiers*, 17(2), 243-259. <https://doi.org/10.1007/s10796-014-9492-7>
- Liao, C.-F., & Weng, Y.-J. (2023). Enabling Space-Aware Service Discovery Model in Home Networks through a Compatible Extension to mDNS/DNS-SD. *Electronics*, 12(18), 3885. <https://doi.org/10.3390/electronics12183885>
- Martín, D., Lamsfus, C., & Alzua-Sorzabal, A. (2016). A cloud-based platform to develop context-aware mobile applications by domain experts [Una plataforma basada en la nube para

- desarrollar aplicaciones móviles sensibles al contexto por parte de expertos en el dominio]. *Computer Standards & Interfaces*, 44, 177-184. <https://doi.org/10.1016/j.csi.2015.08.009>
- Murturi, I., Avasalcai, C., Tsigkanos, C., & Dustdar, S. (2019). Edge-to-Edge Resource Discovery using Metadata Replication. *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, 1-6. <https://doi.org/10.1109/CFEC.2019.8733149>
- Poettering, L., & Llod, T. (2023, diciembre). *Welcome to Avahi* [Org]. avahi. <https://avahi.org>
- Ray, P. P. (2018). A survey on Internet of Things architectures. *Journal of King Saud University - Computer and Information Sciences*, 30(3), 291-319. <https://doi.org/10.1016/j.jksuci.2016.10.003>
- Siljanovski, A., Sehgal, A., & Schonwalder, J. (2014). Service discovery in resource constrained networks using multicast DNS [Descubrimiento de servicios en redes con recursos limitados mediante DNS de multidifusión]. *2014 European Conference on Networks and Communications (EuCNC)*, 1-5. <https://doi.org/10.1109/EuCNC.2014.6882683>
- Stolikj, M., Cuijpers, P. J. L., Lukkien, J. J., & Buchina, N. (2016). Context based service discovery in unmanaged networks using mDNS/DNS-SD. *2016 IEEE International Conference on Consumer Electronics (ICCE)*, 163-165. <https://doi.org/10.1109/ICCE.2016.7430565>
- Trylesinski, M. (s.f). *Uvicorn*. <https://www.uvicorn.org>
- Vermesan, O., Friess, P., Guillemin, P., Gusmeroli, S., Sundmaeker, H., Bassi, A., Jubert, I. S., Mazura, M., Harrison, M., Eisenhauer, M., & Doody, P. (2022). Internet of Things Strategic Research Roadmap [Hoja de ruta de investigación estratégica sobre Internet de las cosas]. En P. Friess & O. Vermesan, *Internet of Things—Global Technological and Societal Trends from Smart Environments and Spaces to Green Ict* (1.<sup>a</sup> ed., pp. 9-52). River Publishers. <https://doi.org/10.1201/9781003338604-2>
- Wei, E. J. Y., & Chan, A. T. S. (2013). CAMPUS: A middleware for automated context-aware adaptation decision making at run time. *Pervasive and Mobile Computing*, 9(1), 35-56. <https://doi.org/10.1016/j.pmcj.2011.10.002>
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer Berlin Heidelberg. <https://doi.org/10.1007/978-3-642-29044-2>