Universidad Zaragoza

1542

Trabajo Fin de Grado

Técnicas de gestión de datos para sistemas de recomendación en entornos distribuidos y dispositivos móviles

Data management techniques for recommendation systems in distributed environments and mobile devices

Autor

Ahmed Karafy Mohib

Director

Sergio Ilarri Artigas

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2024-25

AGRADECIMIENTOS

Quiero expresar mi más sincero agradecimiento a todas las personas que, de una u otra manera, han contribuido a la realización de este Trabajo Fin de Grado.

En primer lugar, a mis padres, por su apoyo incondicional a lo largo de estos años y por enseñarme, con su ejemplo, el valor del esfuerzo y la perseverancia. A mi hermana, por estar siempre presente y acompañarme en este camino, aportando ánimo y cercanía en los momentos más necesarios.

Asimismo, deseo agradecer de manera muy especial a mi tutor de TFG, cuya orientación, paciencia y compromiso han sido fundamentales para el desarrollo de este proyecto. Sin su guía, este trabajo no habría sido posible.

También a mis amigos y compañeros de estudios, que han compartido conmigo experiencias, aprendizajes y buenos momentos durante toda esta etapa universitaria.

Por último, es importante reconocer el respaldo institucional recibido a lo largo de este trabajo, en particular el apoyo del Gobierno de Aragón a través del grupo de investigación *COSMOS* (referencia T64_23R), así como la financiación del proyecto de I+D+i *PID2020-113037RB-I00* (*proyecto NEAT-AMBIENCE*), financiado por *MICIU/AEI/10.13039/501100011033* y liderado por la Universidad de Zaragoza en el que se enmarca este trabajo.

Técnicas de gestión de datos para sistemas de recomendación en entornos distribuidos y dispositivos móviles

RESUMEN

El presente Trabajo Fin de Grado estudia diferentes técnicas de gestión de datos aplicadas al desarrollo de sistemas de recomendación en entornos distribuidos y dispositivos móviles. El objetivo principal es explorar soluciones que permitan desplegar recomendadores embebidos en dispositivos móviles, sin depender de servidores externos, evaluando su viabilidad en términos de privacidad, autonomía y eficiencia.

Para ello, se han analizado dos aplicaciones representativas. En primer lugar, **R-Rules**, un prototipo de motor basado en reglas y activación sensible al contexto, en el que la parte de recomendación se limita a funcionalidades sencillas. El objetivo principal en este trabajo ha sido modernizar su base tecnológica. Esto implicó garantizar la compatibilidad con versiones recientes de Android, mejorar la mantenibilidad del código y sustituir bibliotecas descontinuadas por alternativas activamente mantenidas, facilitando así una mejor experiencia de desarrollo y usuario. Además, se incorporó un módulo de comunicación *peer-to-peer* (P2P) para el intercambio directo de reglas y datos entre dispositivos.

En segundo lugar, se exploró **PASEO**, una aplicación móvil para turismo inteligente que proporciona recomendaciones contextuales y personalizadas. Mientras que su arquitectura original seguía un enfoque cliente-servidor, en este trabajo se investigó la viabilidad de migrar la lógica del sistema al propio dispositivo móvil, complementándola con un módulo *P2P* que permite compartir valoraciones y elementos favoritos entre usuarios, con el fin de mejorar la colaboración y enriquecer las recomendaciones locales.

Este trabajo se enmarca en el proyecto de investigación **NEAT-AMBIENCE**, contribuyendo al avance hacia sistemas de recomendación móviles más autónomos, colaborativos, privados y eficientes.

Índice

1. Introducción y objetivos **1**

1.1. Contexto 1

1.2. Motivación 1

1.3. Objetivos 2

1.4. Alcance del proyecto 3

1.5. Metodología y técnicas empleadas 3

1.6. Estructura de la memoria 4

2. Marco teórico y trabajos relacionados **7**

2.1. Sistemas de recomendación 7

2.2. Sistemas de recomendación móviles 8

2.3. Privacidad y aprendizaje federado 8

2.4. Implementaciones prácticas y prototipos 9

2.5. Comunicación P2P en dispositivos móviles 10

2.6. Síntesis 11

3. R-Rules: migración tecnológica y modernización **13**

3.1. Contexto y motivación 13

3.1.1. Contexto tecnológico del proyecto 14

3.2. Objetivo de la migración 14

3.3. Entorno de desarrollo 14

3.4. Proceso de migración 15

3.4.1. Creación del nuevo proyecto 15

3.4.2. Actualización y sustitución de dependencias 16

3.4.3. Reorganización de la estructura del proyecto 17

3.4.4. Refactorización de componentes 18

3.4.5. Reorganización del módulo Siddhi 19

3.4.6. Migración del módulo nativo de Siddhi a Kotlin 20

3.5. Problemas encontrados durante la migración 20

3.5.1. Problema con dependencia externa 21

3.5.2. Problemas de integración de Siddhi 5.1.30 21
3.6. Comunicación P2P en R-Rules 22
3.6.1. Implementación de la comunicación P2P 23
3.6.2. Flujo de información P2P entre dispositivos 23
3.7. Resultados de la migración 27

4. PASEO: implementación alternativa en el dispositivo 29
4.1. Contexto y motivación 29
4.2. Arquitectura original 29
4.3. Implementación alternativa propuesta 30
4.4. Alcance del trabajo 30
4.5. Objetivos de la implementación alternativa 31
4.6. Arquitectura original del sistema 32
4.6.1. Cliente 32
4.6.2. Servidor 33
4.7. Nueva arquitectura del sistema 34
4.7.1. Migración de la base de datos: de SQLite a Room 34
4.7.2. Sustitución del sistema Virtuoso 36
4.7.3. Acceso a la API SPARQL del Ayuntamiento de Zaragoza 38
4.7.4. Sincronización y actualización de la base de datos 41
4.8. Comunicación P2P en PASEO 43
4.8.1. Implementación de la comunicación P2P 44
4.8.2. Flujo de información P2P entre dispositivos 44
4.9. Evaluación y comparación con la arquitectura previa 46
4.10. Trabajo futuro 49
4.10.1. Exploración de soluciones basadas en redes neuronales 49
4.10.2. Posible reintroducción de SPARQL en escenarios futuros 50
4.11. Resultados de la implementación alternativa 50

5. Evaluación Experimental 53
5.1. Entorno de evaluación y dispositivos utilizados 54
5.2. Evaluación de R-Rules 55
5.2.1. Pruebas de rendimiento del motor Siddhi 56
5.2.2. Comparación de rendimiento Siddhi vs. aplicación completa . . 57
5.2.3. Evaluación del sistema P2P 62
5.3. Evaluación de PASEO 66
5.3.1. Comparativa entre arquitecturas cliente-servidor y local 67

5.3.2. Evaluación del sistema P2P 76
5.4. Discusión general 83
5.5. Conclusiones de la evaluación 84

6. Conclusiones y líneas de trabajo futuro 87
6.1. Conclusiones personales 87
6.2. Conclusiones del proyecto 88
6.3. Limitaciones y trabajo futuro 89

7. Bibliografía 93

Lista de Figuras 97

Lista de Tablas 99

Anexos 100

A. PASEO: esquema relacional de la base de datos 103

B. Implementación del worker de actualización de la base de datos 105

C. Tests de latencia implementados en PASEO 107

D. Medición de métricas de recursos 111

E. Implementaciones del módulo P2P en R-Rules y PASEO 115
E.1. Implementación en R-Rules (Kotlin) 116
E.2. Implementación en PASEO (Java) 120

F. Manual de usuario para despliegue y prueba de R-Rules y PASEO 127
F.1. R-Rules 127
F.1.1. Puesta en marcha del prototipo 128
F.1.2. Recorrido guiado de la aplicación 132
F.1.3. Verificación del funcionamiento de la aplicación 149
F.1.4. Notas adicionales 150
F.2. Despliegue y ejecución de PASEO 150
F.2.1. Puesta en marcha del prototipo 150
F.2.2. Recorrido guiado de la aplicación 151
F.2.3. Verificación del funcionamiento de la aplicación 158
F.2.4. Notas adicionales 159

Capítulo 1

Introducción y objetivos

1.1. Contexto

Los sistemas de recomendación se han consolidado como una de las herramientas más relevantes en la gestión de información personalizada. Su presencia se extiende a múltiples ámbitos, desde el comercio electrónico hasta el ocio digital, ofreciendo a los usuarios contenidos ajustados a sus preferencias y necesidades. En el contexto de los dispositivos móviles, estos sistemas adquieren un papel todavía más significativo, ya que permiten integrar información contextual, como la ubicación geográfica o las interacciones en tiempo real, ampliando así las posibilidades de personalización.

El presente Trabajo Fin de Grado se enmarca en el proyecto de investigación *NEAT-AMBIENCE* [1], que considera entre sus objetivos el estudio de sistemas de recomendación en entornos distribuidos y móviles. El propósito principal es explorar soluciones que permitan desplegar recomendadores de forma autónoma en dispositivos móviles, evitando la dependencia de servidores externos, así como analizar escenarios colaborativos en los que varios dispositivos puedan intercambiar datos o procesamiento.

En este marco se han abordado dos aplicaciones representativas: **R-Rules** [2] y **PASEO** [3, 4], que sirven como casos de estudio para evaluar la viabilidad técnica de diferentes enfoques.

1.2. Motivación

El desarrollo de aplicaciones móviles se enfrenta a varios desafíos asociados a la rápida evolución tecnológica. Uno de ellos es la obsolescencia de las bibliotecas y marcos de trabajo que, con el tiempo, dejan de ser compatibles con las versiones más recientes de los sistemas operativos. Este fue precisamente el caso de **R-Rules**, un prototipo de recomendador sensible al contexto cuya base tecnológica se sustentaba en una versión desactualizada de React Native (0.55.4), junto con dependencias como

NativeBase y componentes Java en la capa nativa. Estas limitaciones dificultaban la evolución del sistema, afectaban a la mantenibilidad del código y restringían las posibilidades de diseño adaptativo. De ahí surgió la motivación de emprender un proceso de modernización que asegurase la compatibilidad con versiones recientes de Android, mejorase la experiencia de desarrollo y facilitase el mantenimiento futuro del software.

Por otro lado, **PASEO** representa un caso de estudio en el ámbito del turismo inteligente, orientado a proporcionar recomendaciones personalizadas a los visitantes de una ciudad. Su arquitectura original seguía un enfoque cliente-servidor, en el que el dispositivo móvil se limitaba a solicitar información a un servidor central encargado de realizar el procesamiento y la lógica de recomendación. Aunque este modelo era funcional, planteaba limitaciones importantes en relación con la privacidad de los usuarios, así como una dependencia constante de la conectividad y los recursos del servidor. Por ello, este trabajo se plantea explorar la viabilidad de migrar la lógica de recomendación al propio dispositivo, con el doble objetivo de reforzar la privacidad y evaluar hasta qué punto los dispositivos móviles actuales cuentan con la capacidad de procesamiento y almacenamiento necesaria para soportar sistemas de recomendación autónomos.

1.3. Objetivos

El objetivo general del presente trabajo es analizar y desarrollar técnicas de gestión de datos aplicadas a sistemas de recomendación móviles y distribuidos, explorando soluciones que permitan tanto la modernización de prototipos existentes como la implementación de arquitecturas autónomas y descentralizadas en dispositivos Android.

De este objetivo general se derivan los siguientes objetivos específicos:

- Modernizar el prototipo **R-Rules**, sustituyendo bibliotecas descontinuadas por alternativas mantenidas, garantizando la compatibilidad con versiones recientes de Android y mejorando la mantenibilidad y experiencia de usuario. Asimismo, incorporar un módulo de comunicación *peer-to-peer* (P2P) que permita el intercambio directo de información y reglas entre dispositivos, potenciando la cooperación entre usuarios sin necesidad de un servidor central.

- Implementar una alternativa para la aplicación **PASEO** siguiendo un enfoque descentralizado, en el que la lógica de recomendación y la gestión de datos se ejecuten directamente en el dispositivo. Además, integrar un sistema de

comunicación *P2P* que permita compartir valoraciones y elementos favoritos entre usuarios de manera segura, reforzando la privacidad y la adaptabilidad del sistema a contextos colaborativos.

– Desarrollar prototipos y pruebas experimentales que permitan analizar el rendimiento, la escalabilidad y las limitaciones de los enfoques propuestos, incluyendo la evaluación del impacto del componente *P2P* en la experiencia de usuario y en el comportamiento global del sistema.

– Comparar las soluciones obtenidas con aproximaciones alternativas descritas en la literatura, evaluando ventajas y desventajas en términos de eficiencia, autonomía, privacidad y colaboración entre usuarios.

En conjunto, estos objetivos orientan el desarrollo del trabajo hacia la búsqueda de soluciones prácticas y comparativas que permitan valorar la idoneidad de arquitecturas distribuidas y descentralizadas en sistemas de recomendación móviles.

1.4. Alcance del proyecto

Este trabajo se centra en el estudio y validación de prototipos, no en el despliegue de sistemas en entornos de producción. El análisis se limita a dispositivos Android, sin considerar otras plataformas móviles. Asimismo, se aborda únicamente el rediseño de dos aplicaciones representativas (R-Rules y PASEO), sin pretender cubrir exhaustivamente todos los posibles sistemas de recomendación móviles existentes. No obstante, los resultados obtenidos permiten extraer conclusiones generales sobre la viabilidad técnica de arquitecturas móviles autónomas y distribuidas.

1.5. Metodología y técnicas empleadas

La metodología seguida combina la revisión de literatura previa con el desarrollo práctico de prototipos experimentales. En el caso de **R-Rules**, el trabajo se centró en la migración tecnológica mediante el uso de herramientas modernas como *React Native* en su versión actualizada, *Gluestack UI* para la interfaz de usuario, y bases de datos locales como *Realm*, manteniendo su lógica original de funcionamiento.

En el caso de **PASEO**, se exploró una implementación alternativa en la que la lógica de recomendación se ejecuta directamente en el dispositivo, evaluando aspectos de rendimiento, almacenamiento local y consumo de recursos.Este segundo caso permitió comparar el enfoque autónomo en dispositivo con arquitecturas cliente-servidor tradicionales, con el fin de analizar sus ventajas y limitaciones.

Además, en ambos prototipos se incorporó y validó un módulo de comunicación *peer-to-peer* (P2P), destinado a posibilitar el intercambio directo de información —reglas, valoraciones o elementos favoritos— entre dispositivos sin necesidad de un servidor intermedio.

1.6. Estructura de la memoria

El resto de la memoria se organiza de la siguiente manera:

- El **Capítulo 2** presenta el estado del arte y los trabajos relacionados en el ámbito de los sistemas de recomendación móviles y distribuidos. Se revisan las principales técnicas existentes, así como diferentes propuestas académicas y soluciones prácticas que sirven de referencia para contextualizar el trabajo realizado.

- El **Capítulo 3** describe en detalle el sistema **R-Rules** y el proceso de migración realizado. Se presenta la arquitectura original, se analizan sus limitaciones y se explica cómo se llevó a cabo la transición hacia una nueva versión adaptada a los requisitos actuales. Asimismo, se detallan las principales decisiones técnicas adoptadas durante la migración, incluyendo la incorporación de un módulo de comunicación *peer-to-peer* (P2P) para el intercambio directo de reglas y datos entre dispositivos.

- El **Capítulo 4** aborda la aplicación **PASEO**, describiendo su arquitectura original y el desarrollo de una implementación alternativa basada en un enfoque autónomo en el propio dispositivo. Este capítulo incluye el diseño de la nueva solución, los aspectos técnicos más relevantes y las decisiones adoptadas para garantizar su funcionamiento en entornos móviles, además de la integración de un componente *P2P* que permite el intercambio de valoraciones y elementos favoritos entre usuarios.

- El **Capítulo 5** presenta la **evaluación experimental** realizada sobre **R-Rules** y **PASEO**. Se detallan las pruebas diseñadas para verificar el correcto funcionamiento de cada uno, así como los resultados obtenidos en distintos escenarios de uso. Además, se analizan aspectos como el rendimiento, la latencia y la estabilidad de los módulos *P2P* integrados en ambos proyectos, evaluando su comportamiento en condiciones reales y comparando sus prestaciones.

- El **Capítulo 6** recoge las conclusiones finales del trabajo, valorando el cumplimiento de los objetivos planteados y destacando las aportaciones más

significativas. Además, se plantean posibles líneas de trabajo futuro orientadas a ampliar y mejorar la propuesta desarrollada.

- Finalmente, la memoria incluye una serie de **Anexos** que complementan la información presentada en los capítulos principales. Entre ellos se encuentran capturas de las aplicaciones desarrolladas (R-Rules y PASEO), el esquema relacional de la base de datos de PASEO, la implementación del `worker` para la actualización periódica de datos, así como los tests de latencia y los scripts empleados para la medición y análisis de métricas de recursos.

Esta estructura permite seguir un hilo conductor claro que parte del contexto teórico, continúa con el desarrollo y modernización de los sistemas estudiados, y concluye con un análisis de resultados y propuestas de mejora.

Capítulo 2

Marco teórico y trabajos relacionados

Este capítulo presenta el estado del arte y los trabajos relacionados en el ámbito de los sistemas de recomendación móviles y distribuidos. En primer lugar, se introducen los fundamentos de los *sistemas de recomendación*, describiendo sus principales enfoques y técnicas. A continuación, se analizan las particularidades de los *sistemas de recomendación móviles*, prestando especial atención al papel del contexto y a las limitaciones inherentes a los dispositivos.

Seguidamente, se aborda la cuestión de la *privacidad de los datos* y el uso del *aprendizaje federado* como alternativa descentralizada para el entrenamiento de modelos. También se revisan diversas *implementaciones prácticas y prototipos* que ilustran la aplicación de estos conceptos, así como los mecanismos de *comunicación P2P en dispositivos móviles*, esenciales para el intercambio directo de información sin un servidor central. Finalmente, se incluye una *síntesis* que resume los principales conceptos tratados y su relación con la propuesta desarrollada.

2.1. Sistemas de recomendación

Los sistemas de recomendación son herramientas diseñadas para ofrecer contenidos personalizados a los usuarios, basándose en sus preferencias, historial de interacción y contexto [5]. Existen diferentes enfoques, entre los que destacan los sistemas colaborativos, basados en contenido y los híbridos. En particular, los sistemas sensibles al contexto incorporan información adicional sobre el entorno del usuario, como ubicación, hora, dispositivo o actividad, con el fin de mejorar la pertinencia de las recomendaciones [5, 6].

La personalización proactiva, que anticipa las necesidades del usuario, y la reactiva, que responde a solicitudes explícitas, son estrategias clave en la experiencia de recomendación [7]. La investigación ha mostrado que la combinación de ambas,

considerando la privacidad del usuario, puede mejorar significativamente la satisfacción y confianza [8, 9].

2.2. Sistemas de recomendación móviles

Los dispositivos móviles plantean desafíos adicionales para los sistemas de recomendación, incluyendo recursos limitados, conectividad intermitente y necesidades estrictas de privacidad. La literatura destaca la importancia de la gestión eficiente de datos locales y la explotación de la información contextual para ofrecer recomendaciones personalizadas [10, 11, 12].

Algunos trabajos se centran en la recomendación basada en trayectorias, especialmente en entornos como museos o recorridos turísticos, donde el contexto geográfico es fundamental [13]. Otros estudios exploran el uso de reglas definidas por el usuario para la recomendación proactiva, maximizando la autonomía del dispositivo y minimizando la dependencia de servidores externos [6].

2.3. Privacidad y aprendizaje federado

La protección de la privacidad es un aspecto crítico en sistemas móviles, especialmente cuando se manejan datos personales o de comportamiento de los usuarios. Los sistemas de recomendación tradicionales suelen requerir la centralización de esta información en servidores remotos, lo que implica riesgos potenciales de filtraciones o usos indebidos de los datos.

El **aprendizaje federado** (*Federated Learning*) surge como una alternativa que permite entrenar modelos de forma colaborativa sin necesidad de transferir los datos fuera del dispositivo del usuario. En este paradigma, cada cliente entrena localmente un modelo con su propia información, y solo se comparten los parámetros actualizados o gradientes con un servidor central encargado de agregarlos y generar un modelo global. De esta manera, la información sensible nunca abandona el dispositivo, lo que reduce significativamente los riesgos de exposición [8, 9].

Este enfoque ha demostrado ser especialmente adecuado en el contexto de **sistemas móviles y distribuidos**, donde los dispositivos presentan capacidades de cómputo heterogéneas y conexiones intermitentes. Además, permite adaptar las recomendaciones a las preferencias locales de cada usuario, mejorando la personalización sin comprometer la privacidad.

En el ámbito de los **recomendadores contextuales**, el aprendizaje federado se ha empleado para combinar información contextual (como ubicación, hora o historial de

actividad) con preferencias personales, manteniendo la confidencialidad de los datos. Este tipo de técnicas resulta relevante para el sistema **PASEO**, ya que abre la posibilidad de incorporar en el futuro modelos de recomendación que aprendan de la interacción de los usuarios de forma descentralizada, preservando su anonimato y cumpliendo con las directrices de privacidad vigentes.

2.4. Implementaciones prácticas y prototipos

Existen múltiples prototipos y aplicaciones que ilustran la implementación de sistemas de recomendación móviles, así como diversas tecnologías de soporte que facilitan su desarrollo:

- **R-Rules**: sistema basado en reglas definidas por el usuario para recomendaciones proactivas, que ha servido como referencia para estudios de migración tecnológica y modernización de entornos React Native [6].

- **PASEO**: aplicación para turismo inteligente que combina datos contextuales y preferencias del usuario para ofrecer recomendaciones sobre puntos de interés, recorridos y eventos. Proyectos recientes exploran implementaciones autónomas en el dispositivo [10, 11].

- Proyectos en GitHub que implementan algoritmos de recomendación para Android utilizando kNN o estrategias híbridas [14, 15].

Además de estos prototipos, es importante destacar algunas tecnologías y enfoques que apoyan la construcción de sistemas de recomendación móviles:

- Herramientas y bibliotecas modernas para desarrollo móvil, como *TensorFlow Lite* [16], *Room* [17], *Realm* [18] y *React Native* [19].

- Uso de RDF en entornos móviles para representar y consultar información semántica [20].

En conjunto, tanto los prototipos como las tecnologías de soporte mencionadas constituyen la base sobre la que se construye la presente investigación, ofreciendo referencias prácticas y herramientas que orientan el diseño y desarrollo de las soluciones propuestas.

2.5. Comunicación P2P en dispositivos móviles

La comunicación *peer-to-peer* (P2P) permite el intercambio directo de información entre dispositivos sin requerir un servidor intermedio ni conexión a Internet. Este enfoque resulta especialmente adecuado para entornos distribuidos, sistemas colaborativos o aplicaciones contextuales en las que los usuarios interactúan físicamente cerca unos de otros.

Entre las principales tecnologías disponibles para la comunicación P2P en dispositivos Android se encuentran *Wi-Fi Direct*, *Bluetooth Low Energy (BLE)*, *WebRTC* y la API *Nearby Connections*, desarrollada por Google. Cada una ofrece un equilibrio distinto entre velocidad, consumo energético, compatibilidad y facilidad de integración.

Tabla 2.1: Comparativa de tecnologías P2P analizadas

Tecnología	**Pros**	**Contras**
Wi-Fi Direct (Wi-Fi P2P)	Alta velocidad de transferencia. Funciona sin Internet. Soporta varios dispositivos en grupo.	Requiere aceptación manual del usuario. APIs complicadas. Compatibilidad irregular entre dispositivos.
Nearby Connections API	Selección automática del mejor transporte (Wi-Fi, Bluetooth, ultrasonido). Conexión y cifrado seguros por defecto. Descubrimiento y conexión automáticos. APIs sencillas y bien documentadas.	Dependencia de Google Play Services (no funciona en dispositivos sin él).
Bluetooth Low Energy (BLE)	Bajo consumo energético. Funciona sin Internet. Adecuado para pequeños mensajes o sincronización ligera.	Distancia muy limitada. Transferencia de datos lenta. APIs más complejas para roles (central/peripheral).
WebRTC (con señalización)	Multiplataforma (Android, iOS, Web). Soporta streaming de voz, vídeo y datos. Cifrado de extremo a extremo integrado.	Necesita servidor de señalización inicial. Configuración más compleja. Menos útil sin conexión a Internet.

Entre estas alternativas, **Wi-Fi Direct** ha demostrado un rendimiento robusto y una buena estabilidad en escenarios de colaboración local. Por ejemplo, en el trabajo *GeoSPRINGS* [21] se empleó esta tecnología para implementar una plataforma móvil basada en agentes con capacidades de comunicación directa entre dispositivos, mostrando su idoneidad para entornos con conectividad limitada. No obstante, la complejidad de sus APIs y la necesidad de confirmación manual en cada

emparejamiento pueden dificultar su uso en aplicaciones con interacciones frecuentes o automatizadas.

Bluetooth Low Energy (BLE) constituye una opción eficiente para el intercambio de pequeños volúmenes de información con bajo consumo energético [22]. Sin embargo, su alcance reducido y su limitada velocidad de transmisión lo hacen menos adecuado para sincronizaciones de datos más pesadas o en escenarios con múltiples dispositivos.

Por su parte, **WebRTC** ofrece una solución multiplataforma y segura para la transmisión de datos, voz o vídeo [23]. Aunque su cifrado de extremo a extremo y su compatibilidad con navegadores lo convierten en una opción atractiva, la necesidad de un servidor de señalización y su dependencia de conexión inicial a Internet restringen su aplicabilidad en contextos puramente desconectados.

Finalmente, la **API Nearby Connections** [24] proporciona un marco de comunicación P2P de alto nivel que abstrae la complejidad del transporte subyacente. La API selecciona automáticamente el canal más adecuado (Wi-Fi, Bluetooth clásico, BLE o incluso ultrasonido) en función de la disponibilidad y las condiciones del entorno, y ofrece descubrimiento, conexión y cifrado de extremo a extremo de forma nativa.

Gracias a esta combinación de simplicidad, seguridad y autonomía, *Nearby Connections* fue la tecnología seleccionada tanto en *R-Rules* como en *PASEO* para implementar sus respectivos módulos P2P, adaptando la integración al lenguaje y arquitectura de cada aplicación (Kotlin en el primer caso y Java en el segundo).

2.6. Síntesis

La revisión del marco teórico y de los trabajos relacionados muestra que los sistemas de recomendación móviles y distribuidos requieren un equilibrio entre personalización efectiva, eficiencia en el uso de recursos y preservación de la privacidad del usuario. Los enfoques basados en contexto permiten adaptar las recomendaciones a la situación específica del usuario, mientras que técnicas como el aprendizaje federado y la ejecución de lógica en el propio dispositivo abordan los desafíos de protección de datos y autonomía.

Los trabajos previos, tanto académicos como implementaciones prácticas, proporcionan metodologías, arquitecturas y prototipos que sirven de referencia para el desarrollo de nuevas soluciones. En particular, los sistemas basados en reglas y los prototipos de recomendación contextual en dispositivos móviles evidencian la viabilidad de modelos autónomos, así como las limitaciones técnicas y oportunidades de optimización.

Este conocimiento fundamenta la aproximación adoptada en el presente trabajo, orientada a explorar implementaciones móviles autónomas y eficientes, capaces de operar sin depender de servidores externos, y establece un marco de comparación frente a arquitecturas más tradicionales cliente-servidor.

Capítulo 3

R-Rules: migración tecnológica y modernización

3.1. Contexto y motivación

En la actual era del *Big Data*, los sistemas de recomendación móviles basados en el contexto juegan un papel fundamental para ayudar tanto a ciudadanos como a turistas a tomar mejores decisiones en su vida cotidiana. En particular, los sistemas de recomendación proactivos son capaces de detectar el momento y el lugar adecuados para ofrecer sugerencias sobre un ítem o actividad específicos, sin necesidad de intervención directa por parte del usuario. Para este fin, se emplean sistemas de tipo *push*, que se apoyan en reglas de contexto para decidir cuándo debe activarse un determinado tipo de recomendación.

No obstante, la literatura recoge aún pocas experiencias sobre la implementación práctica de este tipo de sistemas en dispositivos móviles. Motivados por esta carencia, se diseñó y desarrolló previamente, en el contexto del proyecto NEAT-AMBIENCE, el prototipo **R-Rules (Recommendation Rules!)** [2], cuya principal aportación es la capacidad de lanzar recomendaciones adecuadas en el momento oportuno, de manera automática y sin intervención del usuario.

Con *R-Rules*, el usuario puede activar, desactivar, parametrizar y definir reglas de forma sencilla, lo que permite alcanzar un mayor grado de personalización. Además, el motor de disparo de recomendaciones se ejecuta directamente en el dispositivo móvil, lo que reduce significativamente la necesidad de comunicaciones inalámbricas y contribuye a proteger la privacidad del usuario, dado que los datos de contexto son evaluados localmente en lugar de enviarse a un servidor externo.

3.1.1. Contexto tecnológico del proyecto

El proyecto original de **R-Rules** se basaba en una versión obsoleta de *React Native* (0.55.4), junto con dependencias como *NativeBase* y código Java en la capa nativa. Esta configuración presentaba múltiples limitaciones: incompatibilidades con versiones recientes de Android, dificultades para mantener dependencias y limitaciones en diseño adaptativo. Ante esta situación, se decidió llevar a cabo una migración progresiva del proyecto hacia un entorno actualizado, con el fin de asegurar su sostenibilidad y facilitar el desarrollo futuro.

3.2. Objetivo de la migración

El objetivo principal de la migración fue modernizar la base tecnológica de **R-Rules**, garantizando compatibilidad con versiones recientes de Android, mejorando la mantenibilidad del código y optimizando la experiencia de desarrollo y usuario. De manera específica, se buscó reemplazar bibliotecas descontinuadas por alternativas activamente mantenidas, actualizar dependencias críticas y adaptar la arquitectura de la aplicación a estándares actuales de desarrollo móvil.

Además de estos aspectos técnicos, la migración también tuvo un componente exploratorio: dotar al prototipo de una base más flexible que facilitara la incorporación de nuevas funcionalidades. Gracias a esta actualización fue posible integrar de manera sencilla la comunicación *peer-to-peer* (P2P), ampliando así el alcance experimental de *R-Rules* y permitiendo evaluar escenarios colaborativos de intercambio de recomendaciones.

3.3. Entorno de desarrollo

El proyecto original se desarrolló utilizando versiones considerablemente antiguas de las herramientas principales, lo que generaba múltiples limitaciones de compatibilidad y mantenimiento. En la Tabla 3.1 se recoge la comparación entre el entorno de desarrollo utilizado inicialmente y el actualizado tras la migración.

Como puede observarse, el salto tecnológico ha sido significativo. La elección de Java 17 responde a las recomendaciones oficiales de *React Native* para las versiones más recientes del framework [19].

Tabla 3.1: Comparativa entre el entorno de desarrollo anterior y el actualizado tras la migración

Herramienta	Entorno anterior	Entorno actualizado
Android Studio	4.1.2	2024.3.1 Patch 1
Runtime version	Desconocido	21.0.5+-13047016-b750
Android SDK	Desconocido	API Level 35 (Android 15 Preview)
Node.js	10.24.0	22.14.0
npm	Desconocido	10.9.2
React Native	0.55.4	0.79.1
React Native CLI	2.0.1	18.0.0
Java	1.8	OpenJDK 17.0.14 (Temurin), build 17.0.14+7, release 2025-01-21

3.4. Proceso de migración

Dado el considerable desfase tecnológico del proyecto original, se optó por iniciar un nuevo proyecto desde cero utilizando la versión más reciente de *React Native* disponible en el momento de la migración (v0.79.1). Esta decisión se vio reforzada por la introducción de nuevas funcionalidades esenciales, como el *autolinking*, que facilita la gestión de módulos nativos sin necesidad de enlaces manuales, y por la imposibilidad de actualizar de forma progresiva el proyecto existente debido a errores de compatibilidad al intentar elevar la versión de bibliotecas individuales.

El proceso de migración se estructuró en varias etapas clave con el fin de abordar de manera ordenada los distintos retos técnicos. Cada una de ellas se centró en un aspecto concreto —desde la creación del nuevo proyecto hasta la sustitución de dependencias y la adaptación de componentes nativos—, garantizando así una transición progresiva y controlada.

3.4.1. Creación del nuevo proyecto

Se generó un proyecto limpio mediante el CLI oficial de *React Native* y se configuró el entorno de desarrollo con las versiones más recientes compatibles de Node.js, Java y Android SDK (ver Apartado 3.3). A partir de esta base, se comenzaron a migrar y adaptar los componentes del proyecto original, asegurando que cada módulo funcionara correctamente en el nuevo entorno.

3.4.2. Actualización y sustitución de dependencias

Se instalaron versiones actualizadas de las principales dependencias utilizadas en el proyecto. En algunos casos fue necesario reemplazar bibliotecas obsoletas por alternativas modernas y mantenidas activamente. A continuación se enumeran las principales bibliotecas integradas, agrupadas por funcionalidad:

– **React y React Native:**

- `react: ^19.0.0`
- `react-dom: ^19.0.0`
- `react-native: 0.79.1`

– **Navegación:**

- `@react-navigation/native: ^7.1.6`
- `@react-navigation/native-stack: ^7.3.10`

– **Gestión de gestos y vistas seguras:**

- `react-native-gesture-handler: ^2.25.0`
- `react-native-screens: ^4.10.0`
- `react-native-safe-area-context: ^5.4.0`

– **UI y componentes visuales:**

- Reemplazo de `native-base` por `Gluestack UI`
- `react-native-modal: ^14.0.0-rc.1`
- `react-native-modal-datetime-picker: última versión, requiere @react-native`
- `react-native-vector-icons: ^10.2.0`
- `react-native-star-rating-widget: ^1.9.2`

– **Calendario y mapas:**

- `react-native-calendar-events: ^2.2.0`
- `react-native-maps: ^1.22.1`

– **Base de datos local:**

- `realm: ^20.1.0`

3.4.3. Reorganización de la estructura del proyecto

Junto con la migración tecnológica, se llevó a cabo una profunda reorganización de la estructura del proyecto con el objetivo de mejorar la mantenibilidad, modularidad y claridad del código fuente.

Estructura original (proyecto antiguo)

La estructura original carecía de una clara separación de responsabilidades. Muchos archivos y carpetas coexistían en la raíz del proyecto sin una lógica de agrupamiento clara, lo cual dificultaba la escalabilidad y comprensión del código, especialmente en equipos de trabajo.

```
- App.js
- index.js
- android/
- ios/
- screens/
- siddhi/
- realmSchemas/
- exclusionSets/
- experiments/
- testData/
- event/
- em/
- settings/
- variables.js
```

Estructura nueva (proyecto migrado)

La nueva estructura se diseñó para agrupar archivos y módulos de manera lógica, separando responsabilidades y facilitando la escalabilidad y el mantenimiento. La reorganización incluyó la creación de carpetas específicas para componentes, servicios, eventos, esquemas de base de datos y pruebas, entre otros.

```
- App.js
- index.js
- android/
- ios/
- components/
```

```
- src/
  - background/
  - components/
  - em/
  - events/
  - images/
  - navigation/
  - realmSchemas/
  - screens/
  - services/
  - siddhi/
  - testData/
- jest.config.js
- babel.config.js
- metro.config.js
- gluestack-ui.config.json
- tailwind.config.js
- tsconfig.json
- package.json
```

La nueva estructura adopta un enfoque modular y escalable. Todos los recursos relacionados con la lógica de negocio, pantallas, navegación y servicios se agrupan dentro de una carpeta central `src/`. Se han separado claramente los componentes reutilizables, la lógica de eventos, la gestión de contexto con Siddhi, y los esquemas de Realm. También se introdujeron archivos de configuración para herramientas modernas como Gluestack UI, Tailwind CSS, Prettier, ESLint, y Jest.

Este rediseño de la estructura facilita la colaboración entre desarrolladores (de cara a futuras mejoras), reduce el riesgo de errores por ambigüedad en la ubicación de los archivos, y mejora la escalabilidad del proyecto a futuro.

3.4.4. Refactorización de componentes

Las pantallas originales, basadas en clases y en bibliotecas desactualizadas, fueron refactorizadas utilizando **componentes funcionales** y **hooks** de React. Esta actualización mejora la legibilidad del código, incrementa su modularidad y lo alinea con las prácticas modernas del ecosistema *React Native*.

El uso de hooks permite gestionar el estado y los efectos de manera más clara y concisa, simplificando la integración de la lógica de negocio con la interfaz de usuario

y facilitando futuras ampliaciones o modificaciones del proyecto.

3.4.5. Reorganización del módulo Siddhi

El módulo encargado de la integración con **Siddhi**, el motor de procesamiento de eventos complejos (*CEP*), presentaba inicialmente una gran concentración de responsabilidades. En un solo fichero se gestionaban tanto el ciclo de vida del motor como la escritura dinámica de queries, la generación de reglas y la lógica auxiliar, lo que dificultaba su escalabilidad y mantenibilidad.

Con el objetivo de mejorar la claridad del código y favorecer la extensibilidad futura, se propuso una reestructuración modular de dicho componente, dividiendo sus responsabilidades en submódulos claramente diferenciados. La nueva estructura queda definida de la siguiente manera:

```
siddhi/
- engine.js       # Orquestador principal del motor Siddhi (start, stop, reload)
- index.js        # Punto de entrada/exportación del módulo
- structure.js    # Definiciones estáticas de streams y queries de sistema
                    (intro, end)
- rules/
  - context/
    - index.js    # Función writeAllContextRules
    - generators.js  # Generadores de reglas contextuales
                       (time-based, weather-based etc.)
  - triggering/
    - index.js      # Función writeAllTriggeringRules
    - generator.js  # Generador de reglas de activación simples
  - negatedContext/
    - index.js      # Coordinador de reglas de contexto negadas
    - generator.js  # Generadores específicos para cada tipo de regla negada
```

Funcionalidades por archivo

- **engine.js**: Encapsula el ciclo de vida del motor Siddhi, incluyendo la carga inicial de reglas, el reinicio dinámico y el registro de streams.

- **structure.js**: Contiene la definición base de los flujos *intro*, *end* y otros elementos estructurales comunes a todas las ejecuciones.

– **rules/context/**: Implementa la lógica de generación y escritura de reglas contextuales. Todas las funciones generadoras se agrupan en *generators.js*, mientras que *index.js* se encarga de escribirlas en el motor.

– **rules/triggering/**: Similar a las reglas contextuales, pero orientado a la activación de acciones según condiciones específicas.

– **rules/negatedContext/**: Gestiona aquellas reglas que deben activarse bajo ausencia de contexto (por ejemplo, "no está en casa"), generadas de forma separada para mantener la lógica limpia y separada.

3.4.6. Migración del módulo nativo de Siddhi a Kotlin

El proyecto original contaba con un módulo nativo en Java encargado de la comunicación entre *JavaScript* y el motor *Siddhi* a través del puente de *React Native.* Aunque funcional, dicho módulo presentaba una lógica dispersa y ciertas carencias en cuanto al manejo adecuado de la concurrencia, especialmente en la obtención de resultados desde *Siddhi* tras el envío de eventos.

En la versión anterior del módulo, la obtención del resultado del motor Siddhi dependía de callbacks asincrónicos, lo que requería que el consumidor gestionara manualmente la sincronización sobre el objeto `result`. Esta práctica era propensa a errores en entornos concurrentes, pudiendo generar inconsistencias o race conditions.

En la nueva implementación en Kotlin, se encapsuló la sincronización dentro del módulo mediante mecanismos como `@Synchronized` y `@Volatile`, garantizando un acceso seguro a los datos compartidos y eliminando la necesidad de que el consumidor maneje la concurrencia.

La elección de *Kotlin* frente a *Java* no fue arbitraria. Además de la plena interoperabilidad entre ambos lenguajes en el ecosistema de Android, *Kotlin* ofrece una sintaxis más concisa y expresiva, reduciendo la cantidad de código repetitivo. Asimismo, incorpora de forma nativa características que favorecen la escritura de código más seguro y robusto, como la gestión explícita de nulabilidad y mejores mecanismos para el manejo de concurrencia. Estas ventajas resultaron determinantes para garantizar un diseño más claro, mantenible y menos propenso a errores en la comunicación con el motor *Siddhi.*

3.5. Problemas encontrados durante la migración

Durante el proceso de migración de **R-Rules** surgieron diversos inconvenientes relacionados con la adaptación del código, la gestión de dependencias y la integración

de nuevos componentes. Esta sección recopila los principales problemas detectados, así como las soluciones implementadas para garantizar la compatibilidad y estabilidad del sistema en su nueva versión.

3.5.1. Problema con dependencia externa

Durante el proceso de migración y reconstrucción del entorno surgió una dificultad significativa relacionada con la siguiente librería:

```
org.eclipse.osgi:org.eclipse.osgi.services:3.3.100.v20130513-1956
```

Esta dependencia no se encontraba disponible en los repositorios estándar utilizados por el sistema de construcción. El problema apareció de forma transitiva a través del módulo *Siddhi*, el cual requiere componentes OSGi para el manejo modular y la interacción dinámica entre bundles. En concreto, esta biblioteca proporciona los servicios básicos de OSGi necesarios para garantizar la correcta ejecución de dicho módulo.

Tras varios intentos fallidos de resolverla automáticamente mediante Gradle o Maven, se optó por una solución manual.

En primer lugar, se descargó el fichero `.jar` correspondiente desde el repositorio de Maven Central[1]. Posteriormente, se instaló en el repositorio local de Maven utilizando el siguiente comando:

```
mvn install:install-file \
  -Dfile=org.eclipse.osgi.services-3.3.100.v20130513-1956.jar \
  -DgroupId=org.eclipse.osgi \
  -DartifactId=org.eclipse.osgi.services \
  -Dversion=3.3.100.v20130513-1956 \
  -Dpackaging=jar
```

Este paso resultó imprescindible para que el proyecto pudiera compilar correctamente. A pesar de tratarse de una solución poco ortodoxa, permitió mantener la compatibilidad con el código que dependía de esta versión específica del paquete.

3.5.2. Problemas de integración de Siddhi 5.1.30

Otro de los principales obstáculos encontrados durante la migración fue la integración de *Siddhi* versión 5.1.30 en el entorno Android. Esta versión introduce

[1] https://mvnrepository.com/artifact/org.eclipse.osgi/org.eclipse.osgi.services/3.3.100.v20130513-1956

como dependencia *Log4j*, una biblioteca ampliamente usada en entornos de servidor, pero no compatible con Android de forma nativa.

Se realizaron numerosos intentos para solventar esta incompatibilidad, incluyendo:

- Exclusión manual del paquete *Log4j* en la configuración de Gradle.
- Sustitución del sistema de *logs* por alternativas compatibles.
- Modificación de las `proguard-rules` para eliminar conflictos durante el empaquetado.

Sin embargo, ninguna de estas soluciones resultó completamente funcional en el entorno Android. Como medida final, se optó por recuperar la configuración previa del proyecto original, manteniendo la versión estable 5.1.19 de *Siddhi*, la cual no dependía aún de *Log4j* y funcionaba correctamente en Android.

Esta decisión permitió conservar la funcionalidad del motor de eventos sin comprometer la estabilidad ni introducir dependencias incompatibles.

3.6. Comunicación P2P en R-Rules

En comparación con el prototipo original de *R-Rules*, que operaba de manera aislada en cada dispositivo móvil —evaluando localmente las reglas de contexto y generando recomendaciones individuales—, la nueva versión introduce un mecanismo de comunicación *peer-to-peer* (P2P). Este permite que los terminales móviles no solo ejecuten las reglas internamente, sino que además compartan entre sí las recomendaciones que han sido producidas localmente.

De este modo, cada instancia de *R-Rules* cumple una doble función: actúa como consumidor de recomendaciones basadas en sus propias reglas internas y, al mismo tiempo, como productor y difusor de sugerencias hacia otros dispositivos cercanos. Esta capacidad convierte al sistema en un entorno colaborativo, en el que una recomendación útil para un individuo puede propagarse de manera controlada, potenciando la relevancia de los resultados en contextos sociales, turísticos o de movilidad.

Las principales ventajas de este enfoque incluyen:

- **Colaboración entre usuarios**: las *valoraciones* realizadas por los usuarios sobre los distintos ítems (por ejemplo, puntuaciones o preferencias) pueden compartirse entre dispositivos, permitiendo que el conocimiento colectivo enriquezca el sistema de recomendación.
- **Descentralización**: no es necesario un servidor central para coordinar la difusión de la información, lo que reduce costes y dependencias de infraestructura.

- **Privacidad**: al compartirse únicamente las valoraciones y no los datos de contexto sensibles, se mantiene la filosofía de evaluación local que caracteriza al sistema.
- **Resiliencia**: el sistema se beneficia de la cooperación entre nodos, evitando puntos únicos de fallo y mejorando su tolerancia ante desconexiones o caídas de red.

En conjunto, estas características dotan a *R-Rules* de un modelo de comunicación más robusto y adaptable, que amplía sus posibilidades de aplicación en escenarios reales.

A continuación, en la Sección 3.6.1 se detalla la implementación técnica del módulo de comunicación P2P, describiendo las decisiones de diseño y la tecnología empleada. Posteriormente, en la Sección 3.6.2, se analiza el flujo de información entre dispositivos, incluyendo los mecanismos de descubrimiento, sincronización y propagación de valoraciones.

3.6.1. Implementación de la comunicación P2P

Tal y como se describe en la Sección 2.5, la comunicación *peer-to-peer* del sistema *R-Rules* se implementó mediante un módulo nativo desarrollado en `Kotlin`, integrado directamente en la aplicación móvil. Este módulo utiliza la API *Nearby Connections* de Android, que permite establecer conexiones directas y seguras entre dispositivos cercanos sin necesidad de infraestructura externa.

La integración de la API se llevó a cabo mediante un servicio persistente encargado de gestionar las fases de descubrimiento, conexión y transmisión de datos. Durante el descubrimiento, cada dispositivo anuncia su presencia y detecta otros nodos cercanos que ejecutan la aplicación. Una vez establecida la conexión, el canal P2P permite el intercambio de información contextual y de valoraciones entre usuarios, con cifrado de extremo a extremo proporcionado por la propia biblioteca.

El uso de *Nearby Connections* proporciona así una capa de comunicación local transparente y eficiente, que habilita el funcionamiento colaborativo del sistema incluso en entornos sin conectividad a Internet. Esta integración amplía el modelo de recomendaciones de *R-Rules* desde un enfoque estrictamente individual hacia un paradigma cooperativo y distribuido entre múltiples dispositivos.

3.6.2. Flujo de información P2P entre dispositivos

La comunicación P2P en *R-Rules* se materializa en el intercambio de valoraciones y preferencias asociadas a los ítems que los usuarios consumen o visitan. De este modo,

cada dispositivo actúa simultáneamente como emisor y receptor de esta información, contribuyendo a la generación de recomendaciones locales. El flujo de información sigue un esquema distribuido y autónomo, lo que garantiza que la interacción entre usuarios se lleve a cabo directamente, sin depender de un servidor centralizado.

Descubrimiento y sincronización inicial

Cuando dos dispositivos se descubren mediante la API *Nearby Connections*, intercambian automáticamente las valoraciones realizadas sobre actividades. Este proceso asegura que, desde el primer contacto, ambos nodos disponen de información relevante y filtrada, estableciendo una base común para la propagación posterior de recomendaciones.

Esquema de diseminación y persistencia de información

El modelo de comunicación en *R-Rules* sigue un enfoque oportunista directo: cuando dos dispositivos se encuentran, ambos intercambian las valoraciones que sus respectivos usuarios han realizado sobre los ítems valorados. Cada dispositivo transmite únicamente la información que ha generado localmente, evitando la retransmisión de valoraciones obtenidas de terceros. Este comportamiento garantiza un control natural de la propagación y evita redundancias innecesarias.

Todas las valoraciones se almacenan de forma persistente en la base de datos local del dispositivo, lo que permite que estén siempre disponibles para ser compartidas en futuros encuentros, incluso si la conexión anterior fue interrumpida. Actualmente, el sistema no implementa un control explícito del volumen de datos transmitido, ya que las valoraciones se intercambian de manera completa durante el tiempo que dura la sesión de conexión. No obstante, se contempla como línea de mejora la incorporación de mecanismos de priorización o limitación de volumen, especialmente para entornos con encuentros breves o recursos restringidos.

Este esquema directo y simétrico simplifica la diseminación de información, preservando la autonomía de cada nodo y favoreciendo una propagación eficiente y coherente de las valoraciones entre los usuarios.

Propagación inmediata de nuevas valoraciones

Cada vez que un usuario asigna una valoración a una actividad, la información se transmite de forma inmediata a todos los dispositivos con los que se encuentra conectado, independientemente de que sea positiva o negativa. Este mecanismo refuerza la naturaleza P2P del sistema y proporciona una base de datos más completa para el filtrado colaborativo, ya que las valoraciones bajas también aportan información valiosa

para distinguir las preferencias de los usuarios. De esta forma, las recomendaciones generadas localmente pueden ajustarse con mayor precisión a los gustos reales de cada persona.

Filtrado según preferencias del usuario

Las valoraciones recibidas de otros usuarios no se incorporan directamente al perfil del usuario. Antes de ser utilizadas, se someten a un proceso de filtrado basado en las preferencias individuales de cada persona. Por ejemplo, si un usuario únicamente desea recibir información relacionada con restaurantes, cualquier valoración asociada a actividades de entretenimiento se descarta y no se considera en la generación de recomendaciones. Las valoraciones que superan este filtrado se procesan mediante el sistema de recomendación por defecto (*Mahout*), garantizando que la experiencia resultante se mantenga personalizada y relevante.

Resumen

Este diseño mantiene la filosofía descentralizada y autónoma de *R-Rules*, incrementando la riqueza de las recomendaciones a través de la colaboración directa entre dispositivos, sin comprometer la privacidad ni la personalización de cada usuario. De este modo, la comunicación P2P no solo transmite información, sino que la integra de manera coherente con las preferencias locales, preservando la efectividad del motor de recomendaciones original (ver Figura 3.1; para detalles de implementación ver Anexo E.1).

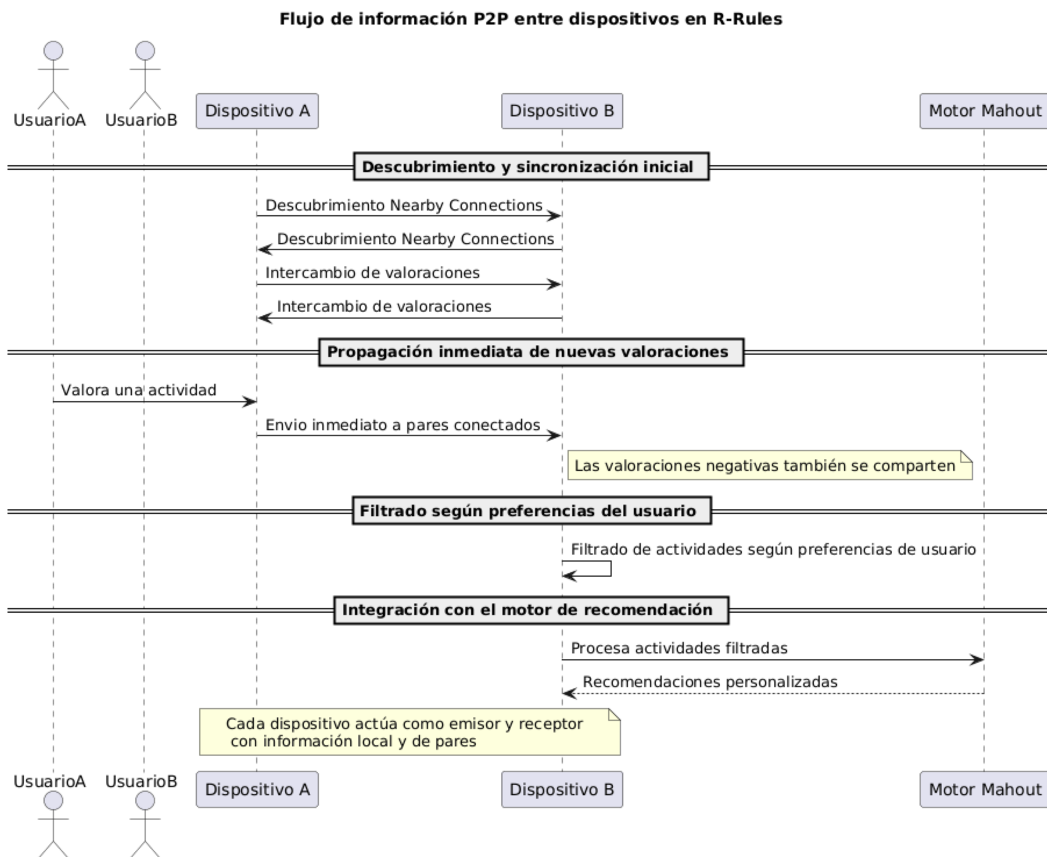


Figura 3.1: Diagrama de secuencia del flujo de información P2P entre dispositivos en *R-Rules*.

3.7. Resultados de la migración

La migración del proyecto Android ha representado una oportunidad para modernizar de manera integral tanto el entorno de desarrollo como la arquitectura del código. Partiendo de una versión obsoleta y con dependencias desactualizadas, se ha reconstruido el proyecto desde cero utilizando herramientas actuales y alineadas con las mejores prácticas del ecosistema de *React Native* y Android.

Durante el proceso, se adoptaron decisiones clave orientadas a mejorar la mantenibilidad y escalabilidad del sistema. Entre ellas destacan la reestructuración modular del código —con especial énfasis en el módulo de *Siddhi*— y la sustitución de bibliotecas descontinuadas, como *NativeBase*, por soluciones modernas y activamente mantenidas, como *Gluestack UI*.

Gracias a esta modernización, fue posible incorporar nuevas funcionalidades que anteriormente no eran viables, siendo la más relevante la integración de un módulo de comunicación *peer-to-peer* (P2P). Este módulo permite que los dispositivos móviles intercambien recomendaciones de forma directa y segura mediante la API *Nearby Connections*, transformando R-Rules desde un modelo individual hacia uno cooperativo entre usuarios.

A lo largo de la migración surgieron diversos desafíos técnicos —como la incompatibilidad de *Siddhi* 5.1.30 con Android debido a su dependencia de *Log4j*, o la necesidad de configurar manualmente ciertas dependencias nativas— que fueron abordados mediante soluciones adaptadas al nuevo entorno. A pesar de estas dificultades, la migración se completó con éxito: el sistema conserva la funcionalidad original y, al mismo tiempo, se encuentra preparado para su evolución futura, incluyendo la capacidad de comunicación P2P y la ampliación de sus reglas de contexto.

La evaluación de la aplicación migrada, así como el análisis de su comportamiento en distintos escenarios de uso, se detalla en el Capítulo 5. Los resultados obtenidos confirman que el nuevo entorno mejora tanto el rendimiento como la mantenibilidad del código, a la vez que ofrece una base sólida para el desarrollo de futuras funcionalidades.

En definitiva, el proceso de migración no solo ha supuesto una actualización tecnológica, sino también una evolución conceptual del proyecto R-Rules. La incorporación del módulo P2P refuerza su carácter colaborativo y proactivo, potenciando la personalización y la relevancia de las recomendaciones en contextos sociales y turísticos.

Para facilitar la comprensión y el uso del prototipo desarrollado, se ha elaborado un manual de usuario detallado que incluye tanto la guía de instalación como un recorrido completo por la aplicación, con capturas de pantalla y explicaciones de cada

funcionalidad. Este manual proporciona instrucciones paso a paso que permiten a cualquier usuario replicar la instalación, configurar el sistema y explorar las distintas secciones de la aplicación de manera autónoma.

Se recomienda consultar el manual de usuario de *R-Rules* (Sección F.1) para obtener información completa sobre la instalación, configuración y utilización de la aplicación, así como para entender en detalle el flujo de trabajo de las recomendaciones, la gestión de reglas de activación (*Triggering Rules*), conjuntos de exclusión (*Exclusion Sets*) y reglas de contexto (*Context Rules*).

En la Tabla 3.2 se resumen las principales tecnologías y herramientas empleadas en el desarrollo de R-Rules, junto con su función dentro del proyecto.

Tabla 3.2: Resumen de tecnologías utilizadas en R-Rules

Categoría	**Tecnología / Herramienta**	**Función en el proyecto**
Lenguaje	JavaScript	Desarrollo del frontend de R-Rules
Lenguaje	Kotlin	Implementación del módulo nativo de Siddhi
Framework	React Native	Construcción de la interfaz de usuario
Base de datos	Realm	Gestión local de datos en la aplicación
Virtualización	VMware	Creación de entornos de prueba y desarrollo aislados
Herramientas	Android Studio / Gradle / Git	Compilación, desarrollo y control de versiones

Capítulo 4

PASEO: implementación alternativa en el dispositivo

4.1. Contexto y motivación

En el marco del turismo inteligente y la personalización de contenidos, este capítulo presenta una **implementación alternativa de PASEO 2.0** (*Profile generation And content Suggestion on E-TOurism*), una aplicación móvil previamente desarrollada en un proyecto anterior. En esta versión se ha llevado a cabo una reconstrucción del sistema con tecnologías actuales, orientada a ofrecer a los turistas sugerencias e información adaptada a sus preferencias personales. El objetivo principal de esta aplicación es asistir a los usuarios durante sus visitas, proporcionándoles recomendaciones contextuales sobre puntos de interés (PIs), recorridos personalizados o eventos temporales, todo ello de forma proactiva y dinámica.

Para esta implementación se ha partido de un prototipo de *PASEO* procedente de proyectos previos: *PASEO 2.0* [4] y *PASEO 1.0* [3].

PASEO está concebida para funcionar en dispositivos móviles como teléfonos inteligentes o tabletas, aprovechando su capacidad de movilidad, conectividad y personalización. La aplicación incorpora un sistema de recomendación basado en el contexto del usuario, que tiene en cuenta tanto su ubicación geográfica como su historial de interacciones. Entre las fuentes de información que el sistema puede utilizar se incluyen las valoraciones que el usuario realiza sobre los elementos visitados, las fotografías que toma, sus interacciones sociales y otros factores contextuales que puedan inferirse.

4.2. Arquitectura original

La versión inicial de PASEO seguía un enfoque cliente-servidor, en el que la lógica de recomendación y gran parte del procesamiento de datos se llevaban a cabo en un

servidor externo, mientras que el cliente (la aplicación en el dispositivo) se limitaba a mostrar la información y enviar solicitudes. Este modelo ofrecía simplicidad en el desarrollo, pero generaba ciertas limitaciones en términos de dependencia de red, consumo energético y exposición de datos personales.

4.3. Implementación alternativa propuesta

En lugar de mantener el esquema original, se propone explorar una **implementación alternativa** en la que la lógica del sistema se ejecute directamente en el dispositivo del usuario. Esta aproximación persigue dos objetivos principales:

- **Privacidad**: al mantener los datos y el procesamiento en el propio dispositivo, se minimiza la necesidad de enviar información sensible a servidores externos, reforzando así la protección de la información personal del usuario.

- **Viabilidad técnica**: se busca evaluar hasta qué punto resulta factible implementar sistemas de recomendación completos y contextuales que funcionen de manera autónoma en un dispositivo Android. Esta exploración implica analizar aspectos como el rendimiento, la gestión de almacenamiento local y las capacidades de computación en dispositivos móviles actuales.

Estos dos ejes orientan el desarrollo hacia soluciones que no solo protejan mejor la información del usuario, sino que también permitan valorar la factibilidad real de ejecutar sistemas de recomendación avanzados en entornos móviles.

En este contexto, surge posteriormente la incorporación de un **módulo de comunicación P2P** (*peer-to-peer*), que extiende este principio de descentralización permitiendo el intercambio directo de información entre dispositivos cercanos. Con este componente, cada usuario no solo ejecuta el sistema de recomendación de manera local, sino que además puede compartir y recibir datos relevantes —como valoraciones o actividades favoritas— sin necesidad de conectarse a un servidor central. Esta ampliación refuerza la privacidad y la autonomía del sistema, y abre la puerta a escenarios de colaboración distribuida entre usuarios.

4.4. Alcance del trabajo

Este capítulo aborda el estudio de **dos arquitecturas distintas** para la aplicación *PASEO*:

- La arquitectura **original cliente-servidor**, en la cual la lógica de recomendación se ejecuta en un servidor externo y el dispositivo actúa principalmente como cliente.

- Una **implementación alternativa local**, en la que toda la lógica de recomendación se traslada al propio dispositivo Android, con el objetivo de explorar su viabilidad técnica y sus implicaciones en términos de privacidad, rendimiento y autonomía.

En esta segunda arquitectura se ha incorporado además un **módulo de comunicación P2P**, que permite el intercambio directo de información entre dispositivos cercanos sin necesidad de conexión a Internet. Este componente amplía el alcance del estudio al introducir un modelo **colaborativo descentralizado**, donde los usuarios pueden compartir datos relevantes de manera local, reforzando la privacidad y la autonomía del sistema.

Es importante subrayar que estas dos versiones se han desarrollado y analizado de forma **independiente**, sin coexistir dentro de una misma aplicación. El trabajo, por tanto, no plantea un sistema híbrido o conmutador entre arquitecturas, sino una comparación experimental entre dos enfoques diferentes.

De este modo, los resultados obtenidos permiten identificar las ventajas y limitaciones de cada alternativa, aportando una visión crítica sobre la idoneidad de trasladar el procesamiento desde el servidor al dispositivo en el contexto de aplicaciones turísticas inteligentes.

4.5. Objetivos de la implementación alternativa

El objetivo principal de esta fase del proyecto es desarrollar una **implementación alternativa de PASEO** basada en una arquitectura completamente local, en la que toda la lógica de negocio, el sistema de recomendación y el almacenamiento de datos residan exclusivamente en el dispositivo Android del usuario.

Esta versión no solo busca eliminar la dependencia del servidor, sino también explorar mecanismos de **colaboración directa entre dispositivos** mediante comunicación P2P, ampliando las posibilidades de funcionamiento autónomo sin conexión a Internet.

De forma más concreta, esta implementación persigue los siguientes objetivos:

- **Eliminar la dependencia del servidor externo**: trasladando toda la lógica que anteriormente se ejecutaba en el servidor (consultas, inferencias, generación

de recomendaciones) al entorno móvil. Esto incluye transformar peticiones HTTP y respuestas JSON en llamadas internas y acceso directo a bases de datos locales.

- **Garantizar la persistencia de datos en el dispositivo**: sustituyendo las soluciones de almacenamiento remoto por bases de datos locales usando tecnologías como Room o Realm, de modo que el sistema pueda operar de forma completamente autónoma, incluso sin conexión a Internet.

- **Incorporar un módulo de comunicación P2P**: permitiendo que los dispositivos cercanos intercambien información relevante (por ejemplo, actividades o recomendaciones) sin depender de infraestructura externa. Este componente extiende la autonomía del sistema hacia un modelo **colaborativo y descentralizado**.

- **Preservar la funcionalidad existente**: asegurando que las características fundamentales del sistema (como la generación de recomendaciones contextuales o el manejo de datos del perfil del usuario) sigan funcionando correctamente en la versión local.

- **Mejorar la privacidad del usuario**: al mantener todos los datos en el dispositivo, se evita su transmisión a servidores externos, lo que representa una mejora significativa desde el punto de vista de la privacidad y la protección de datos personales.

En conjunto, estos objetivos permiten sentar las bases para una aplicación más ligera, privada, robusta y sostenible, capaz no solo de operar de forma autónoma, sino también de **colaborar localmente con otros dispositivos** para enriquecer la experiencia del usuario sin comprometer su privacidad. Este capítulo se centrará en documentar esta **implementación local de PASEO**, analizando sus implicaciones técnicas y comparándola con la arquitectura original cliente-servidor.

4.6. Arquitectura original del sistema

La versión inicial del sistema **PASEO** seguía una arquitectura cliente-servidor tradicional, en la que las responsabilidades estaban claramente distribuidas entre los dos extremos de la aplicación.

4.6.1. Cliente

El cliente consistía en una aplicación Android que actuaba principalmente como interfaz de usuario. Su función era presentar al usuario final la información recibida

desde el servidor y permitirle interactuar con el sistema a través de diferentes pantallas: consulta de puntos de interés, generación de recorridos personalizados, acceso a eventos temporales, valoraciones, etc. Esta parte no contenía lógica de negocio significativa, sino que servía como puente de entrada y salida para la información.

4.6.2. Servidor

El servidor concentraba el núcleo funcional de la aplicación, siendo responsable de los siguientes aspectos:

- **Gestión de usuarios:** registro, autenticación e identificación.
- **Generación de recomendaciones:** incluyendo lógica contextual basada en posición, historial del usuario y eventos.
- **Persistencia de datos:** mediante una base de datos SQLite local al servidor.
- **Comunicación con fuentes externas:**
  - **Virtuoso:** un motor de base de datos RDF utilizado para realizar búsquedas semánticas basadas en palabras clave, explotando relaciones entre conceptos.
  - **API SPARQL de datos abiertos de Zaragoza:** `http://datos.zaragoza.es/sparql`, que permite la consulta de información en tiempo real sobre elementos urbanos y turísticos relevantes, como eventos, monumentos, actividades culturales, etc.

La comunicación entre el cliente y el servidor se realizaba mediante peticiones HTTP (principalmente de tipo POST), enviando y recibiendo información en formato JSON. El servidor respondía con los resultados procesados y empaquetados para que el cliente los presentara al usuario.

Esta estructura permitía una separación clara de responsabilidades, pero presentaba ciertas limitaciones para los objetivos actuales del proyecto. Entre ellas, destacaba la posible exposición de datos del usuario. Concretamente, la aplicación manejaba información sensible como la ubicación geográfica en tiempo real, así como las valoraciones (puntuaciones con estrellas) y los sitios añadidos a favoritos. Estos datos, aunque necesarios para la personalización de recomendaciones, podían llegar a perfilar de manera detallada las preferencias y hábitos del usuario, lo que hacía imprescindible replantear su gestión bajo criterios de seguridad y privacidad más estrictos.

4.7. Nueva arquitectura del sistema

Con el objetivo de maximizar la privacidad del usuario, simplificar la infraestructura y permitir el funcionamiento offline, el sistema **PASEO** se ha rediseñado para funcionar completamente dentro del propio dispositivo Android, eliminando así la necesidad de depender de un servidor externo para la mayor parte de su funcionamiento.

En esta nueva arquitectura, toda la lógica de negocio, el sistema de recomendación, la persistencia de datos y la gestión del perfil del usuario residen en el dispositivo. El acceso a la *API SPARQL de datos abiertos de Zaragoza* (`https://datos.zaragoza.es/sparql`) ya no constituye la fuente primaria de información en cada ejecución, sino que se utiliza de manera complementaria. Los datos sobre puntos de interés y eventos turísticos se almacenan de forma local en una base de datos **Room**, lo que permite realizar búsquedas rápidas y reducir la dependencia de la red.

La *API SPARQL* se consulta únicamente en dos casos:

- Cuando un término no existe en la base de datos local.
- Durante tareas periódicas de actualización para mantener la información sincronizada.

Este nuevo enfoque permite que la aplicación funcione de forma autónoma en la mayoría de escenarios, garantizando además un mayor control de los datos por parte del usuario.

4.7.1. Migración de la base de datos: de SQLite a Room

Uno de los primeros pasos en esta transformación fue la sustitución del sistema de persistencia basado en **SQLite** (usado en el servidor) por una base de datos **Room**, completamente integrada en el entorno Android.

Durante el análisis de alternativas, se valoró el uso de **Realm**, una base de datos móvil moderna diseñada para ofrecer acceso rápido a datos estructurados, con sincronización automática y sin necesidad de escribir consultas SQL. Realm es muy popular en entornos móviles debido a su sencillez, su rendimiento y su capacidad para trabajar bien con estructuras complejas y reactivas.

Sin embargo, para este proyecto se optó finalmente por **Room** por las siguientes razones:

- **Compatibilidad directa con SQLite**: dado que la base de datos original ya estaba implementada en SQLite, la migración a Room resulta más directa y menos costosa en términos de refactorización. Room actúa como una capa de abstracción sobre SQLite, permitiendo mantener el modelo relacional existente.

– **Integración oficial con Android Jetpack**: Room es parte del conjunto oficial de bibliotecas de Android, lo que garantiza compatibilidad, mantenimiento y mejor integración con componentes como *LiveData*, *ViewModel* y *Coroutines*.

– **Mayor control de las consultas**: Room permite escribir SQL personalizado cuando es necesario, algo útil en esta migración, ya que muchas de las consultas ya estaban definidas en el sistema original y no era necesario redefinir toda la lógica de acceso a datos.

El uso de Room aporta varios beneficios a la arquitectura:

– Persistencia local segura y estructurada.

– Simplificación de pruebas y depuración gracias a su integración con herramientas de desarrollo de Android.

– Eliminación del servidor como intermediario para operaciones de lectura/escritura, ya que las consultas se realizan directamente sobre la base de datos local.

El esquema completo de la base de datos puede consultarse en el Anexo A.

Aunque en principio también podría haberse optado por utilizar directamente SQLite en el dispositivo, esta alternativa presenta ciertos inconvenientes en comparación con el uso de Room. Si bien SQLite constituye el motor de base de datos subyacente y ofrece control completo mediante consultas SQL, trabajar directamente sobre él requiere implementar manualmente aspectos clave como la gestión de conexiones, el mapeo entre tablas y objetos de dominio, o la verificación de la validez de las consultas.

Room, en cambio, ofrece una capa de abstracción que mantiene la compatibilidad total con SQLite pero aporta beneficios significativos:

– **Seguridad en tiempo de compilación:** Room valida las consultas SQL en fase de compilación, reduciendo errores que solo se detectarían en tiempo de ejecución si se usara SQLite directamente.

– **Productividad y legibilidad:** mediante anotaciones y DAOs, el acceso a datos se define de forma declarativa, evitando gran parte del código repetitivo asociado al manejo manual de cursores y transacciones.

– **Integración con el ecosistema Android:** al ser parte de Jetpack, Room se integra de manera nativa con LiveData, Flow, Coroutines y ViewModel,

simplificando la implementación de patrones arquitectónicos modernos (MVVM, Clean Architecture).

En este sentido, aunque SQLite directo podría haber resultado suficiente para la persistencia local, el uso de Room garantiza un desarrollo más robusto, mantenible y alineado con las buenas prácticas recomendadas por Android, manteniendo además la eficiencia del motor SQLite que utiliza internamente.

4.7.2. Sustitución del sistema Virtuoso

En la arquitectura original, el sistema utilizaba **Virtuoso**, un motor de base de datos RDF con soporte completo para SPARQL, para realizar consultas semánticas basadas en relaciones entre términos. Esta herramienta permitía implementar búsquedas por palabras clave con una semántica rica, ofreciendo funcionalidades como la inferencia de conceptos relacionados o la navegación por jerarquías de ontologías.

No obstante, en el contexto de la nueva arquitectura local basada exclusivamente en Android, Virtuoso no es una opción viable, por las siguientes razones:

- No existe una versión oficial de Virtuoso compatible con Android.
- El tamaño del motor y su complejidad lo hacen inviable para ejecutarse dentro de un dispositivo móvil sin acceso a un servidor externo.
- Su uso requiere recursos y bibliotecas nativas que no están disponibles en el entorno Android.

Ante esta situación, se exploraron diferentes alternativas que ofrecieran capacidades similares en un entorno local y con soporte para Android. La Tabla 4.1 resume las opciones consideradas.

Tras realizar la comparación entre las distintas soluciones disponibles (Tabla 4.1), se observa que, aunque todas permiten cierto grado de gestión semántica en entornos móviles, existen diferencias notables en cuanto a rendimiento, complejidad y grado de integración con la lógica del sistema.

En particular, destaca *Androjena*, un port parcial de Jena para Android que permite ejecutar consultas SPARQL y trabajar con RDF de manera nativa en el dispositivo. Su principal ventaja radica en aprovechar una librería consolidada en el ámbito de la Web Semántica, con un tamaño moderado (2–5 MB) y un rendimiento aceptable para conjuntos de datos pequeños o medianos. Sin embargo, algunos estudios [25] han mostrado que, si bien es factible realizar razonamiento semántico en dispositivos

Tabla 4.1: Comparativa de alternativas exploradas para sustituir el motor Virtuoso en la arquitectura local de PASEO

Opción	**Soporte SPARQL**	**Rendimiento en Android**	**Tamaño / Complejidad**	**Integración con lógica existente**
Sesame / RDF4J	Sí	Media-alta (problemas en Android)	JARs pesados, problemas con SPI	Alta
Androjena (Jena)	Sí	Media (buena para datos pequeños/ medianos)	2–5 MB, puerto parcial para Android	Media-alta
TriplePlace	No (consultas propias)	Muy buena, indexación eficiente	Ligera, pensada para Android	Alta
SQLite / Room	No (solo SQL)	Muy buena, nativa	Ligero, sin dependencias externas	Muy alta
Mobile RDF	Sí	Media (consultas SPARQL locales)	Tamaño moderado, librería menos madura	Media (útil si se explotan capacidades semánticas)

móviles, las limitaciones de memoria y procesamiento reducen considerablemente su eficiencia frente a soluciones más ligeras.

En este proyecto, dado que las consultas requeridas son simples y pueden implementarse eficientemente mediante SQL, se optó por no introducir la complejidad adicional que supondría integrar Androjena. A partir de esta comparación, se concluyó que la opción más adecuada era prescindir completamente de SPARQL y modelar la lógica semántica con SQLite a través de Room. La decisión se basó en las siguientes consideraciones clave:

- **Simplicidad y ligereza:** al tratarse de una aplicación móvil, es prioritario minimizar el tamaño del APK, reducir la carga de dependencias externas y evitar problemas de compatibilidad. Room cumple con todos estos requisitos, siendo parte del ecosistema oficial de Android.

- **Lógica RDF sencilla:** el uso de Virtuoso en este proyecto se limitaba a búsquedas básicas por palabras clave y relaciones directas entre términos. Estas relaciones pueden replicarse en Room mediante tablas intermedias y lógica SQL personalizada.

- **Mayor control y personalización:** al no depender de un motor RDF externo, se dispone de control total sobre el modelo de datos y las consultas, lo que facilita ajustes, pruebas y mantenimiento.

- **Limitaciones de otras alternativas:** también se evaluaron soluciones como Mobile RDF [20], pero se descartaron debido a su falta de actualización (la última versión disponible data de 2008), lo que plantea problemas de compatibilidad, mantenimiento y seguridad en el contexto actual de desarrollo móvil.

En consecuencia, se ha implementado una nueva base de datos local adicional en Room que contiene las estructuras necesarias para reemplazar el comportamiento anterior de Virtuoso. Esto ha permitido mantener la funcionalidad original, reduciendo al mismo tiempo la complejidad y mejorando la portabilidad del sistema.

4.7.3. Acceso a la API SPARQL del Ayuntamiento de Zaragoza

Uno de los componentes esenciales del sistema PASEO es su capacidad de obtener datos turísticos actualizados desde la plataforma de datos abiertos del Ayuntamiento de Zaragoza, accesible a través de su endpoint SPARQL: `https://datos.zaragoza.es/sparql`

Esta API proporciona acceso en tiempo real a información estructurada sobre puntos de interés, eventos, actividades culturales, servicios públicos y otros recursos de interés.

En la versión original del sistema, implementada en el servidor, las consultas SPARQL se realizaban mediante la librería Jena, una herramienta consolidada para el manejo de RDF y SPARQL en Java. No obstante, Jena no resulta viable en entornos móviles debido a su tamaño, a la presencia de dependencias nativas y a su uso intensivo de mecanismos de reflexión y servicios (SPI), lo que impide su integración en Android.

En la nueva arquitectura, donde la mayor parte de la lógica y los datos se gestionan localmente mediante una base de datos en el dispositivo, el acceso al endpoint SPARQL se mantiene únicamente como fuente complementaria. El sistema consulta en primer lugar la base de datos local y, solo en caso de que un término de búsqueda no esté disponible, se lanza una consulta a la API.

Asimismo, se ha incorporado un mecanismo de actualización periódica en segundo plano que sincroniza la base de datos local con el endpoint. De esta forma, se garantiza que la información almacenada en el dispositivo se mantenga actualizada sin depender exclusivamente de las peticiones en tiempo real.

La base de datos local de Puntos de Interés (PIs) del Ayuntamiento de Zaragoza contiene 33.505 registros (fecha de consulta 8 de septiembre de 2025), según la consulta SPARQL mostrada en la Tabla 4.2.

Tabla 4.2: Consulta SPARQL para obtener el número total de PIs en el endpoint del Ayuntamiento de Zaragoza

```
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX v: <http://www.w3.org/2006/vcard/ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT (COUNT(DISTINCT ?uri) AS ?total)
WHERE {
  ?uri a ?s;
       rdfs:label ?nombre.

  OPTIONAL { ?uri rdfs:comment ?comment. }
  OPTIONAL { ?uri v:photo ?photo. }
}
```

Cada registro contiene información como nombre, descripción, categoría y, en muchos casos, una imagen representativa del PI (URL). Estimando un tamaño promedio de 500–600 bytes por registro, el almacenamiento total requerido asciende aproximadamente a 17–20 MB. Esta estrategia de almacenamiento local permite un acceso rápido y consultas offline para el usuario, garantizando eficiencia en dispositivos Android actuales. Dado que los teléfonos modernos suelen disponer de varios gigabytes de almacenamiento, un consumo de 20 MB se considera insignificante y no supone un inconveniente para el usuario. En futuras versiones, podrían implementarse técnicas de compresión o carga selectiva de PIs para optimizar aún más el uso de memoria en escenarios con bases de datos más extensas.

La descripción detallada de este proceso se desarrolla en el Apartado 4.7.4.

Implementación en Android con OkHttp3

Para implementar este acceso al endpoint en Android se ha optado por una solución ligera y totalmente compatible con el entorno móvil: realizar las consultas SPARQL mediante peticiones HTTP utilizando la librería `OkHttp3`.

Aunque Android proporciona clases nativas como `HttpURLConnection`, `OkHttp3` ofrece ventajas relevantes en este contexto:

- **Simplicidad y claridad de código:** su API es más moderna e intuitiva, reduciendo la complejidad del manejo de peticiones.

- **Gestión robusta de conexiones:** incorpora de forma nativa mecanismos de recuperación, reintentos automáticos y gestión de caché.
- **Flexibilidad:** permite enviar consultas SPARQL como texto plano en el cuerpo de la petición HTTP y procesar directamente las respuestas en formato JSON.

Además, mientras que la versión original del sistema utilizaba el endpoint `http://datos.zaragoza.es/sparql`, en la nueva arquitectura se accede a su versión segura `https://datos.zaragoza.es/sparql`.

El uso de HTTPS no es un detalle menor, sino una decisión fundamentada en tres aspectos clave:

- **Cifrado de datos en tránsito:** garantiza que las consultas y respuestas no puedan ser interceptadas ni manipuladas por terceros.
- **Compatibilidad con Android:** desde Android 9 (API 28), el uso de HTTP sin cifrado está restringido por defecto, lo que convierte HTTPS en el estándar recomendado.
- **Buenas prácticas en servidores públicos:** muchos endpoints, incluido el de Zaragoza, aplican políticas de seguridad que bloquean o redirigen conexiones no seguras.

Flujo de acceso a la API SPARQL

El flujo implementado en la nueva arquitectura para obtener datos desde el endpoint SPARQL es el siguiente:

1. **Búsqueda en la base de datos local:** la aplicación consulta primero la base de datos local (Room/SQLite) para obtener la información solicitada.
2. **Verificación de resultados:**
   - Si los datos están disponibles en la base de datos local, se devuelven directamente al usuario.
   - Si para algún término buscado no existen resultados (*miss*), se procede a consultar la API SPARQL en tiempo real.
     - **Construcción dinámica de la consulta SPARQL:** se genera la query SPARQL como una cadena de texto adaptada al término buscado.
     - **Petición HTTP:** la consulta se envía al endpoint mediante la librería `OkHttp3`.

- **Procesamiento de la respuesta:** la aplicación recibe el resultado en formato JSON, lo parsea y extrae los datos relevantes.
- **Actualización de la base de datos local:** los nuevos datos obtenidos del endpoint se insertan en la base de datos local, de modo que estén disponibles para futuras consultas sin necesidad de recurrir de nuevo al SPARQL.

3. **Entrega al usuario:** finalmente, los datos procesados se muestran en la aplicación.

Esta solución logra un equilibrio entre eficiencia y actualización en tiempo real. El uso prioritario de la base de datos local garantiza un acceso rápido, ligero y funcional incluso en escenarios sin conexión, mientras que la integración con el endpoint SPARQL permite complementar la información con datos actualizados cuando sea necesario. De este modo, se evita la dependencia de bibliotecas pesadas o servidores intermedios, se mantiene un nivel de seguridad adecuado y se ofrece al usuario una experiencia fluida, consistente y confiable.

El diagrama de la Figura 4.1 ilustra de manera gráfica el flujo descrito anteriormente.

4.7.4. Sincronización y actualización de la base de datos

Con el objetivo de asegurar que la base de datos local refleje la información más reciente disponible en el endpoint SPARQL de Zaragoza, se ha implementado un sistema de actualización periódica en segundo plano.

El uso de este mecanismo ofrece varias ventajas:

- **Automatización:** la actualización se realiza sin intervención manual.
- **Fiabilidad:** garantiza la ejecución aunque la aplicación no esté activa.
- **Consistencia:** mantiene la base de datos local alineada con la información externa.
- **Eficiencia:** el intervalo semanal reduce el consumo de red y batería, equilibrando actualización y rendimiento.

De esta manera, el sistema combina consultas bajo demanda con procesos de sincronización periódica, logrando un equilibrio entre inmediatez, disponibilidad y eficiencia en el uso de recursos.

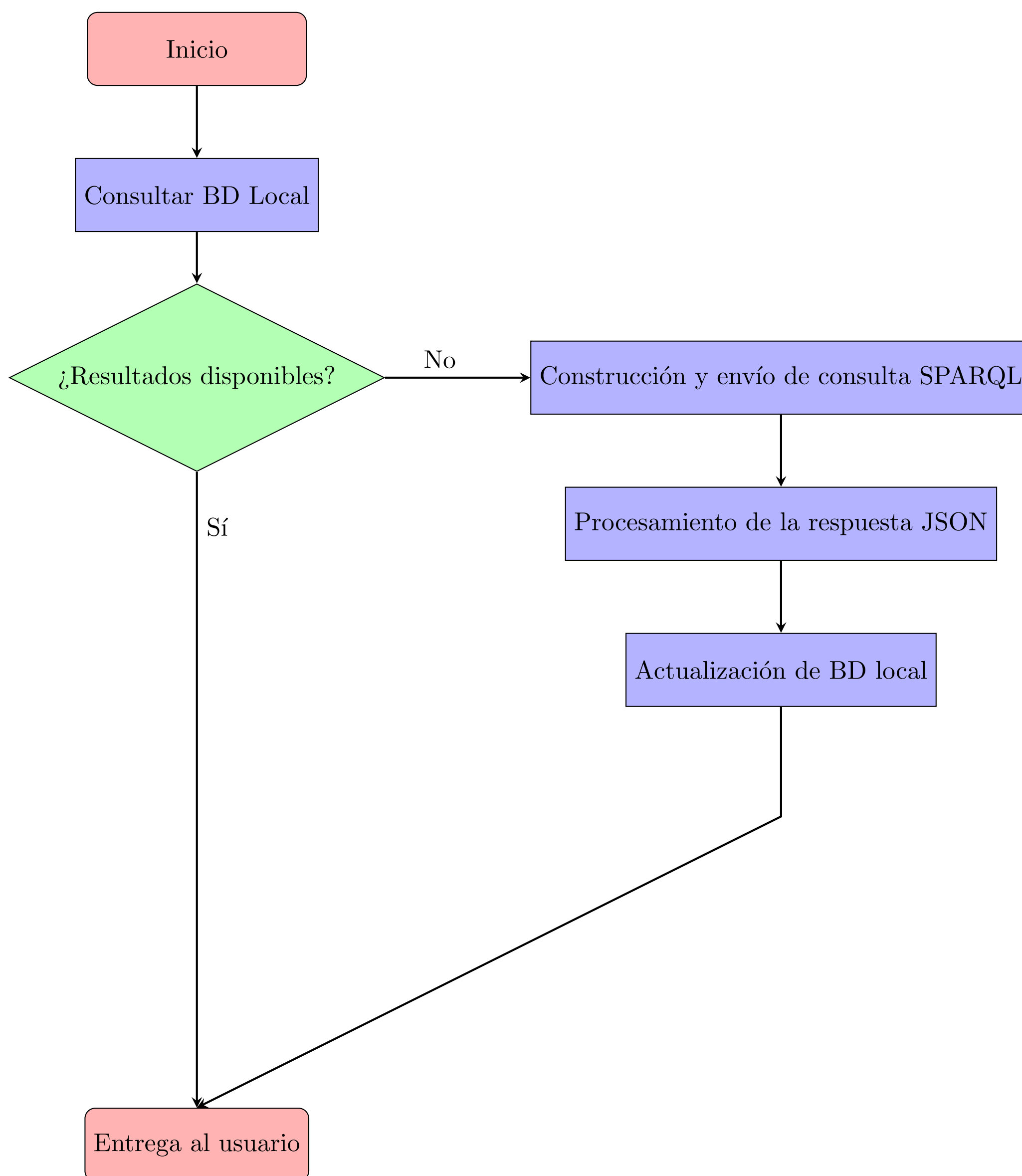


Figura 4.1: Flujo de acceso a la API SPARQL con integración local

En el caso de información asociada a eventos con un intervalo temporal definido, como conciertos o actividades culturales, el mecanismo de actualización asegura que los registros obsoletos no permanezcan en la base de datos local. Para mantener la consistencia y reflejar con fidelidad la información disponible en el endpoint SPARQL, periódicamente se realiza una re-sincronización completa de la base de datos: los eventos que ya han pasado se eliminan automáticamente y se insertan los registros actualizados o nuevos. De este modo, la base de datos local siempre contiene únicamente información vigente, garantizando que el usuario reciba recomendaciones y notificaciones precisas y actualizadas.

El código correspondiente a la implementación del *worker* puede consultarse en el Anexo B.

4.8. Comunicación P2P en PASEO

En contraste con las versiones iniciales del sistema, donde las actividades y registros de los usuarios se gestionaban de manera aislada en cada dispositivo, *PASEO* incorpora un mecanismo de comunicación *peer-to-peer* (P2P) que permite el intercambio directo de información entre terminales cercanos. Gracias a este modelo distribuido, cada dispositivo puede compartir sus interacciones y recibir las de otros, fomentando una red colaborativa de conocimiento local sin depender de un servidor central.

Cada instancia de *PASEO* desempeña así una doble función: actúa como nodo emisor, difundiendo sus propias actividades y observaciones, y como nodo receptor, integrando la información recibida de otros usuarios para enriquecer su contexto local. Este proceso favorece la construcción colectiva del entorno compartido, manteniendo la autonomía de cada dispositivo.

Las principales ventajas de este enfoque son:

- **Colaboración entre usuarios**: las *interacciones* y experiencias registradas por un usuario pueden ser difundidas hacia otros dispositivos, mejorando la capacidad del sistema para generar sugerencias o rutas más relevantes.

- **Descentralización**: la comunicación se establece de forma directa entre dispositivos, sin la necesidad de un servidor central, reduciendo la dependencia de infraestructura externa.

- **Privacidad**: se intercambian únicamente datos anonimizados de interacción, preservando la información contextual sensible en el dispositivo del usuario.

- **Resiliencia**: la red P2P permite mantener la funcionalidad incluso ante desconexiones parciales, gracias a la replicación y persistencia distribuida de la información.

En conjunto, este modelo dota a *PASEO* de una arquitectura más abierta, colaborativa y tolerante a fallos, alineada con la filosofía de autonomía y descentralización que caracteriza al sistema.

En las siguientes secciones se describe con detalle la estructura interna del módulo P2P. En primer lugar, la Sección 4.8.1 presenta la implementación técnica del sistema, incluyendo la gestión de conexiones, el formato de los mensajes y los mecanismos de sincronización local. A continuación, la Sección 4.8.2 profundiza en el flujo de información entre dispositivos, abordando el descubrimiento, la diseminación de actividades y el filtrado según términos de interés.

4.8.1. Implementación de la comunicación P2P

La implementación de la comunicación *peer-to-peer* en *PASEO* se basa en un modelo distribuido de intercambio de información entre dispositivos Android cercanos. Para ello, el sistema emplea un servicio de descubrimiento y conexión directa que permite detectar otros nodos activos en el entorno y establecer canales seguros de transmisión de datos.

Cada dispositivo mantiene una cola local de eventos e interacciones recientes, que son empaquetadas en mensajes de sincronización. Estos mensajes se transmiten de forma asíncrona a los nodos vecinos detectados, garantizando que la información se propague de manera gradual y controlada sin requerir conexión a un servidor central. A su vez, los datos recibidos son validados e integrados en la base de datos local, evitando duplicidades mediante el uso de identificadores únicos asociados a cada interacción.

El módulo P2P está diseñado para funcionar de forma transparente al usuario, ejecutándose en segundo plano y reanudando las transmisiones pendientes cuando la conectividad entre nodos vuelve a estar disponible. Este enfoque permite una comunicación continua, resiliente y eficiente en entornos de movilidad, preservando la autonomía de cada dispositivo dentro de la red distribuida.

4.8.2. Flujo de información P2P entre dispositivos

La comunicación P2P en *PASEO* se materializa en el intercambio de valoraciones y preferencias asociadas a los ítems que los usuarios consumen o visitan. De este modo, cada dispositivo actúa simultáneamente como emisor y receptor de esta información, contribuyendo a la generación de recomendaciones locales. El flujo de información sigue un esquema distribuido y autónomo, lo que garantiza que la interacción entre usuarios se lleve a cabo directamente, sin depender de un servidor centralizado.

Descubrimiento y sincronización inicial

Cuando dos dispositivos se descubren mediante la API *Nearby Connections*, se realiza un intercambio inicial de información sobre actividades que el usuario haya valoradow, así como aquellas que haya marcado como favoritas. Este proceso asegura que ambos nodos compartan desde el primer momento un conjunto de datos relevantes y seleccionados, lo que sienta las bases para una red colaborativa de recomendaciones locales.

Esquema de diseminación y persistencia de información

El modelo de comunicación en *PASEO* sigue un enfoque oportunista directo: cuando dos dispositivos se encuentran, ambos intercambian las valoraciones que sus respectivos usuarios han realizado sobre los ítems valorados. Cada dispositivo transmite únicamente la información que ha generado localmente, evitando la retransmisión de valoraciones obtenidas de terceros. Este comportamiento garantiza un control natural de la propagación y evita redundancias innecesarias.

Todas las valoraciones se almacenan de forma persistente en la base de datos local del dispositivo, lo que permite que estén siempre disponibles para ser compartidas en futuros encuentros, incluso si la conexión anterior fue interrumpida. Actualmente, el sistema no implementa un control explícito del volumen de datos transmitido, ya que las valoraciones se intercambian de manera completa durante el tiempo que dura la sesión de conexión. No obstante, se contempla como línea de mejora la incorporación de mecanismos de priorización o limitación de volumen, especialmente para entornos con encuentros breves o recursos restringidos.

Propagación inmediata de nuevas interacciones

Cada vez que un usuario asigna una valoración a una actividad, o añade una actividad a su lista de favoritas, la información se transmite automáticamente a todos los dispositivos con los que mantiene conexión. Este mecanismo garantiza que las actualizaciones se difundan de forma continua y en tiempo real, permitiendo que cada nodo actúe como un punto activo dentro de una red de conocimiento compartido.

Filtrado según términos de interés

Las recomendaciones recibidas no se integran directamente, sino que se someten a un proceso de filtrado basado en los términos de interés definidos por el usuario (por ejemplo, *"parque"*, *"museo"* o *"monumento"*). Este filtrado depende del tipo de recomendación configurado en los ajustes de la aplicación, aplicando el método correspondiente según las preferencias individuales. De este modo, cada usuario recibe únicamente las actividades que se ajustan a su perfil, manteniendo la relevancia y personalización del sistema.

Resumen

Este diseño conserva la filosofía descentralizada y colaborativa de *Paseo*, potenciando la riqueza del ecosistema de recomendaciones mediante la difusión directa entre dispositivos. La combinación de intercambio inmediato y filtrado por términos

garantiza una experiencia dinámica, contextual y ajustada a los intereses locales del usuario. En conjunto, la comunicación P2P no solo distribuye información, sino que la adapta inteligentemente a las preferencias y configuraciones de cada participante (ver Figura 4.2; para detalles de implementación, consultar el Anexo E.2).

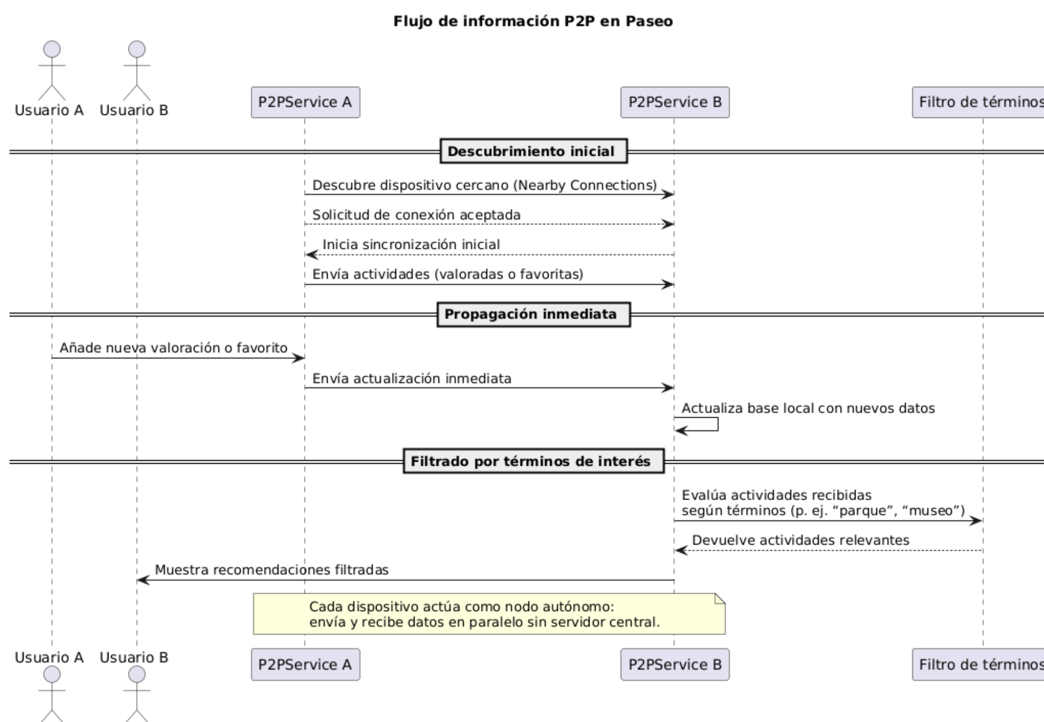


Figura 4.2: Diagrama de secuencia del flujo de información P2P entre dispositivos en *Paseo*.

4.9. Evaluación y comparación con la arquitectura previa

La migración llevada a cabo en este trabajo debe entenderse como un cambio de paradigma en la arquitectura del sistema. En la versión original de *PASEO 2.0*, la aplicación móvil actuaba como un cliente ligero cuya función principal era mostrar la información obtenida desde un servidor central. Este servidor concentraba la lógica de recomendación y se encargaba de ejecutar las consultas SPARQL sobre el endpoint remoto, procesar los resultados y enviarlos de vuelta al cliente.

Este enfoque presentaba ventajas evidentes: los datos estaban siempre actualizados en tiempo real y el dispositivo del usuario apenas soportaba carga de cómputo. Sin embargo, también implicaba una fuerte dependencia de la conectividad a Internet, un mayor tiempo de respuesta debido a la doble capa de red y procesamiento, así como la exposición de información sensible (ubicación, historial de visitas, valoraciones) a un

servidor externo.

La nueva arquitectura propuesta elimina la figura del *backend* y traslada la lógica de recomendación directamente al dispositivo Android. En este modelo, la aplicación accede de manera prioritaria a una base de datos local gestionada mediante Room, de modo que la mayoría de interacciones del usuario se resuelven de manera inmediata y sin necesidad de conexión.

El uso de SPARQL no desaparece, pero queda relegado a un papel secundario y puntual: se utiliza únicamente en el arranque inicial para poblar la base de datos local, como mecanismo de respaldo en caso de que una consulta no pueda resolverse en la información almacenada (*miss* de caché), y de forma periódica para sincronizar y actualizar los datos (por ejemplo, con una frecuencia semanal).

En otras palabras, mientras que en la versión cliente-servidor SPARQL era el mecanismo principal para todas las consultas de recomendación, en la arquitectura local su papel queda reducido a tareas de inicialización y sincronización.

A esta descentralización se añade ahora un componente de comunicación P2P, que permite el intercambio directo de información contextual y de recomendaciones entre dispositivos cercanos, sin necesidad de un servidor intermedio. Este mecanismo amplía las capacidades del sistema en escenarios sin conectividad y fomenta la colaboración distribuida entre usuarios, ya que cada nodo puede actuar tanto como consumidor como proveedor de conocimiento contextual.

Gracias a este enfoque híbrido —local y P2P—, *PASEO* logra una mayor resiliencia, privacidad y escalabilidad: los datos personales permanecen en el dispositivo, el sistema sigue funcionando incluso sin conexión a Internet, y la red de usuarios contribuye colectivamente al enriquecimiento del conocimiento disponible.

Como síntesis de lo expuesto, en la Tabla 4.3 se presenta una comparación entre la arquitectura cliente-servidor original de *PASEO 2.0* y la nueva arquitectura local en dispositivo.

Este cambio ofrece varias ventajas significativas. La primera es la mejora en **privacidad**, ya que los datos sensibles del usuario permanecen en el propio dispositivo. En el contexto de *PASEO 2.0*, estos datos incluyen la ubicación GPS (que permite trazar desplazamientos), el historial de visitas (que revela intereses culturales y hábitos de movilidad) y las valoraciones sobre recursos turísticos (que construyen un perfil de preferencias). En el modelo cliente-servidor toda esta información se transmitía al *backend*, quedando expuesta a riesgos de interceptación o uso indebido. En la arquitectura local, en cambio, la información permanece en el dispositivo, lo que reduce el riesgo de filtraciones y refuerza el principio de *privacidad por diseño*.

La segunda ventaja es la **autonomía de uso**: al no depender de una conexión

Aspecto	**Arquitectura cliente-servidor (PASEO 2.0 original)**	**Arquitectura local en dispositivo (nueva)**
Uso de SPARQL	Consultas en tiempo real al endpoint remoto. Potencial para consultas complejas y extensibles.	Uso inicial para poblar la BD, en sincronizaciones periódicas o en fallos de caché. Consultas diarias resueltas sin SPARQL.
Latencia	Mayor, por dependencia de red y doble procesamiento (servidor + cliente).	Menor, al resolverse en la BD local.
Privacidad	Datos sensibles (ubicación, historial, valoraciones) enviados a servidor externo.	Datos permanecen en el dispositivo, reduciendo exposición.
Dependencia de conectividad	Alta: no funciona sin Internet.	Baja: funciona en modo *offline*, solo necesita conexión esporádica.
Consumo de recursos	Bajo en el dispositivo, alto en servidor.	Mayor uso de CPU, memoria y batería del móvil.
Coherencia de datos	Actualización en tiempo real y centralizada.	Actualización periódica (riesgo de desincronización).
Escalabilidad	Servidor central puede optimizar y escalar para muchos usuarios.	Escalabilidad depende de la capacidad de cada dispositivo.

Tabla 4.3: Comparación entre la arquitectura cliente-servidor original y la nueva arquitectura local en dispositivo

continua, la aplicación puede seguir proporcionando recomendaciones incluso en entornos turísticos con conectividad limitada. A ello se suma la mejora en la experiencia de usuario, pues la latencia se reduce notablemente al resolver la mayoría de las consultas de manera local. Además, la incorporación de un mecanismo **P2P** permite el intercambio directo de información contextual y recomendaciones entre dispositivos cercanos, lo que amplía la funcionalidad del sistema en ausencia de conexión y refuerza la idea de colaboración descentralizada entre usuarios.

No obstante, este nuevo modelo también plantea desafíos: la coherencia de los datos ya no es inmediata, sino que depende de procesos de actualización periódicos, y el procesamiento recae ahora sobre los recursos del dispositivo, lo que requiere optimización en términos de uso de memoria y energía.

En resumen, puede afirmarse que la migración representa un **equilibrio distinto**

entre inmediatez y autonomía. Mientras que la arquitectura cliente-servidor garantizaba la actualización continua y centralizada, lo hacía a costa de la privacidad y la dependencia de la red. La arquitectura actual, en cambio, prioriza la seguridad, la robustez y la usabilidad en escenarios reales de turismo, reduciendo la frecuencia de acceso a SPARQL hasta lo estrictamente necesario. El resultado es un sistema más cercano a la idea de un asistente autónomo, capaz de seguir funcionando en condiciones adversas de conectividad y más respetuoso con la privacidad de los usuarios, aunque con el coste de asumir que ciertos datos puedan no estar en tiempo real.

La evaluación detallada de este cambio arquitectónico se presenta en la sección **5.3.1** dentro del capítulo **5**.

4.10. Trabajo futuro

Aunque el sistema migrado funciona correctamente y permite ejecutar toda la lógica de recomendación desde Android, existen múltiples líneas de trabajo futuro que podrían mejorar o ampliar las capacidades de la aplicación. En las siguientes subsecciones se describen algunas de las más relevantes.

4.10.1. Exploración de soluciones basadas en redes neuronales

En este contexto, podría explorarse el uso de modelos más avanzados con redes neuronales ejecutadas directamente en el dispositivo. En particular, *TensorFlow Lite* [16], una versión ligera del framework TensorFlow, puede integrarse en Android para realizar inferencias de predicción localmente. Esto permitiría, por ejemplo, personalizar recomendaciones mediante un modelo entrenado con datos de interacción previos del usuario, sin depender de un servidor.

Aunque esta línea no ha sido implementada en el presente trabajo, se han investigado opciones viables, como los siguientes ejemplos:

- **Recommendation System for Android con kNN** (GitHub) [14].

- **Movie Recommender App** (GitHub) [15].

Estos proyectos ilustran cómo puede llevarse a cabo un sistema de recomendación personalizado y ligero, ejecutado íntegramente en Android, lo cual encaja con la filosofía de descentralización que ha inspirado esta implementación alternativa.

4.10.2. Posible reintroducción de SPARQL en escenarios futuros

La decisión de no utilizar SPARQL en este proyecto estuvo plenamente justificada, dado que las capacidades semánticas requeridas eran limitadas y podían replicarse mediante consultas SQL simples en Room. Sin embargo, esta elección podría suponer una restricción en fases futuras de evolución del sistema. En caso de que se planteen mejoras que requieran un razonamiento semántico más avanzado, la integración de un motor compatible con SPARQL volvería a ser de interés.

Entre los posibles escenarios donde SPARQL resultaría ventajoso se incluyen consultas más complejas que combinen múltiples relaciones RDF, razonamiento sobre ontologías o jerarquías de conceptos e interoperabilidad con sistemas externos que utilicen datos enlazados o grafos RDF.

En ese contexto, se podría valorar la incorporación de bibliotecas como *Androjena* u otras soluciones actualizadas para Android, teniendo en cuenta la literatura existente sobre razonamiento semántico en dispositivos móviles [25]. De esta forma, se mantendría la portabilidad y ligereza lograda en este proyecto, sin renunciar a la posibilidad de enriquecer la dimensión semántica del sistema en fases posteriores.

Es importante señalar que, en la implementación actual de PASEO, no se ha considerado razonamiento semántico avanzado sobre jerarquías o subclases. Por ejemplo, si un usuario está interesado en espectáculos, la aplicación no infiere automáticamente que conciertos, cines u otras actividades relacionadas podrían ser de interés. No obstante, en escenarios futuros donde se requiera este tipo de inferencia semántica, la reintroducción de un motor compatible con SPARQL podría resultar ventajosa, permitiendo enriquecer la experiencia del usuario mediante recomendaciones más completas y contextualizadas.

4.11. Resultados de la implementación alternativa

En este capítulo se ha abordado en profundidad el análisis comparativo de las arquitecturas locales y cliente-servidor en la aplicación *PASEO*, evaluando su rendimiento, consumo de recursos y experiencia de usuario. El objetivo principal ha sido comprobar la viabilidad de ejecutar toda la lógica de recomendación directamente en un dispositivo Android, manteniendo la funcionalidad de la versión original y adaptándose a las particularidades del entorno móvil.

Durante el estudio se han identificado y resuelto desafíos técnicos relevantes, como la sustitución de tecnologías no compatibles con Android (por ejemplo, Jena o Mahout) por alternativas viables como OkHttp3 para consultas SPARQL o Room como sistema

de persistencia. Se ha demostrado que es posible integrar de manera eficiente tanto la lógica de negocio como el acceso a datos y las funcionalidades de recomendación en Android, reduciendo la dependencia de servidores externos y mejorando la portabilidad de la aplicación.

Asimismo, se ha preservado la conexión con fuentes externas esenciales, como la API abierta del Ayuntamiento de Zaragoza, garantizando la seguridad en la comunicación mediante HTTPS y la eficiencia en el acceso a los datos remotos.

Los resultados de las pruebas de latencia y consumo de recursos muestran que la ejecución local ofrece tiempos de respuesta significativamente más rápidos, mientras que la arquitectura cliente-servidor reduce la carga en el dispositivo a costa de una mayor latencia. Esta información proporciona una base sólida para tomar decisiones sobre la arquitectura más adecuada según el escenario de uso y los recursos disponibles.

Finalmente, la incorporación de un componente **P2P** supone un paso adicional hacia la descentralización completa del sistema. Este nuevo mecanismo permite la cooperación directa entre dispositivos, posibilitando el intercambio de información contextual o de recomendaciones sin necesidad de un servidor central. De esta forma, *PASEO* evoluciona hacia un modelo distribuido más resiliente, capaz de mantener su funcionalidad incluso sin conectividad y de aprovechar el conocimiento colectivo de los usuarios.

En definitiva, el análisis realizado evidencia la viabilidad técnica y práctica de consolidar en un único dispositivo móvil un sistema originalmente distribuido, manteniendo su funcionalidad y abriendo nuevas posibilidades para futuras ampliaciones, como la integración de modelos de recomendación basados en aprendizaje automático, el filtrado colaborativo distribuido o la reintroducción de capacidades semánticas mediante SPARQL. Esta sección, por tanto, proporciona conclusiones claras sobre el rendimiento, la eficiencia y la capacidad evolutiva de *PASEO* en distintos escenarios, sentando las bases para su próxima generación tecnológica.

La evaluación completa del sistema *PASEO* —incluyendo la comparativa entre arquitecturas cliente-servidor y local, así como las pruebas sobre el módulo *peer-to-peer*— se presenta en la Sección 5.3 del Capítulo 5.

Para facilitar la comprensión y el uso del prototipo *PASEO*, se ha elaborado un manual de usuario detallado que incluye un recorrido completo por la aplicación, con capturas de pantalla y explicaciones de cada funcionalidad. Este manual proporciona instrucciones paso a paso que permiten a cualquier usuario iniciar sesión, explorar los puntos de interés recomendados, valorar y añadir lugares a favoritos, así como configurar las preferencias de la aplicación de manera autónoma.

Se recomienda consultar el manual de usuario de *PASEO* (Sección F.2) para obtener

información completa sobre la navegación dentro de la aplicación, el uso del menú lateral, la gestión de puntos de interés, ajustes del sistema y las funcionalidades de valoración y favoritos, asegurando así un aprovechamiento óptimo del prototipo.

En la Tabla 4.4 se presentan las principales tecnologías y herramientas empleadas en el desarrollo de PASEO.

Tabla 4.4: Resumen de tecnologías utilizadas en PASEO

Categoría	**Tecnología / Herramienta**	**Función en el proyecto**
Lenguaje	Java	Backend y frontend de PASEO
Framework	N/A	No se utilizó framework específico para el frontend
Base de datos	Room	Almacenamiento local de datos en PASEO
Virtualización	VMware	Entornos de prueba y desarrollo aislados
Herramientas	Android Studio / Gradle / Git	Desarrollo, compilación y control de versiones

Capítulo 5

Evaluación Experimental

En este capítulo se presentan las pruebas experimentales diseñadas para validar el correcto funcionamiento y el rendimiento de los sistemas desarrollados a lo largo de este trabajo. El objetivo principal es comprobar que las modificaciones introducidas —tanto a nivel de arquitectura como de implementación— cumplen los requisitos de eficiencia, estabilidad y usabilidad definidos en los capítulos anteriores.

Antes de abordar los resultados obtenidos, se describe el *entorno de evaluación* utilizado, incluyendo los dispositivos, configuraciones y herramientas empleadas durante las pruebas. Este apartado inicial proporciona el contexto necesario para interpretar adecuadamente los experimentos y asegurar la reproducibilidad de los resultados.

A continuación, la evaluación se estructura en dos bloques principales. En primer lugar, se analiza el comportamiento del sistema *R-Rules* tras su migración, evaluando el rendimiento tanto del motor de procesamiento de eventos *Siddhi* como de la aplicación completa y del módulo de comunicación *peer-to-peer* (P2P). Para ello, se llevaron a cabo tres conjuntos de pruebas: (i) mediciones de uso de CPU y memoria en el motor *Siddhi*, (ii) comparación de rendimiento entre *Siddhi* y la aplicación integrada, y (iii) análisis del módulo P2P, evaluando la latencia de transmisión, el impacto del volumen de datos y el comportamiento del sistema ante desconexiones y reconexiones. Estas pruebas permiten determinar la eficiencia global del sistema y su capacidad para mantener una operación estable bajo distintas condiciones de carga y conectividad.

En segundo lugar, se presenta la evaluación del sistema *PASEO*, cuyo propósito es analizar el rendimiento y la viabilidad de la arquitectura local propuesta frente al modelo cliente-servidor original. Para ello, se llevó a cabo una comparativa entre ambas configuraciones midiendo el consumo de CPU y memoria, con el objetivo de determinar el impacto que la descentralización tiene sobre la eficiencia del sistema. Además, se evaluó el módulo *peer-to-peer* (P2P) integrado en esta arquitectura, orientando las pruebas a validar su contribución a la calidad de las recomendaciones generadas. Con

este fin, se establecieron los siguientes objetivos específicos: (i) evaluar la precisión de las recomendaciones tras el intercambio de información P2P, (ii) analizar la cobertura personalizada que aportan los datos procedentes de otros pares, y (iii) cuantificar el control de irrelevancia, verificando que la difusión no afecta negativamente a la personalización de los resultados. En conjunto, estas pruebas permiten valorar la eficacia del enfoque distribuido tanto en términos de rendimiento como de calidad de la experiencia ofrecida al usuario.

Los resultados obtenidos en ambas fases de evaluación permiten valorar la efectividad de las decisiones de diseño adoptadas y la viabilidad de la arquitectura propuesta.

5.1. Entorno de evaluación y dispositivos utilizados

Las pruebas experimentales se llevaron a cabo en distintos dispositivos Android con el objetivo de evaluar el rendimiento, consumo de recursos y estabilidad de los sistemas *R-Rules* y *PASEO* en condiciones reales de ejecución. Se seleccionaron terminales representativos de diferentes gamas de hardware para garantizar la validez de los resultados.

- **Google Pixel 9 Pro**

  - Procesador: Google Tensor G4 A520 (1.92 GHz)
  - Memoria RAM: 12 GB
  - Versión de Android: 15

  Este dispositivo se empleó en las pruebas de rendimiento y migración de *R-Rules*. Su entorno homogéneo permitió obtener medidas estables y reproducibles, sirviendo como referencia base para comparar el comportamiento del motor Siddhi frente a la aplicación completa.

- **Xiaomi Redmi Note 10S**

  - Procesador: MediaTek Helio G95 (octa-core, hasta 2,05 GHz)
  - Memoria RAM: 8 GB
  - Almacenamiento interno: 128 GB
  - Versión de Android: 13
  - Capa de personalización: MIUI 14

Se utilizó en las pruebas del módulo P2P de *R-Rules* y también en la evaluación de *PASEO*. Representa un dispositivo de gama media que permite analizar la eficiencia y la escalabilidad de las soluciones propuestas en entornos de recursos limitados.

– **Vivo Y36**

  - Procesador: Snapdragon 680 (octa-core, hasta 2,4 GHz)
  - Memoria RAM: 16 GB (8 + 8 GB dinámicos)
  - Almacenamiento interno: 256 GB
  - Versión de Android: 15
  - Capa de personalización: Funtouch OS 15

  Este dispositivo, de gama media-alta, se empleó tanto en las pruebas del módulo P2P como en las de *PASEO*, proporcionando un punto de comparación para analizar la influencia de los recursos de hardware en el rendimiento del sistema distribuido.

En todos los casos, los experimentos se realizaron bajo condiciones controladas: sin procesos en segundo plano y con la aplicación recién iniciada, asegurando así la repetibilidad de los resultados.

Además de los dispositivos móviles, algunos experimentos requerían la ejecución de componentes del sistema en un servidor externo. Para ello se empleó siempre el mismo equipo: un portátil **MSI GL75 Leopard 10SEK-261XES** (Intel Core i7-10750H, 16 GB de RAM, SSD de 1 TB y GPU NVIDIA RTX 2060) con *Ubuntu 22.04.5* ejecutado de forma nativa.

Este equipo actuó como servidor en dos contextos diferenciados: (i) en *R-Rules*, alojando el *Environment Manager* durante las pruebas que requerían su ejecución externa al dispositivo; y (ii) en *PASEO*, como nodo servidor en la configuración cliente–servidor evaluada. Su uso permitió estudiar el comportamiento de ambos sistemas cuando parte del procesamiento se delega a un servidor externo, complementando así los experimentos realizados íntegramente en los terminales Android.

5.2. Evaluación de R-Rules

En esta sección se presenta la evaluación experimental realizada sobre el sistema *R-Rules*. El objetivo principal de esta fase fue validar el correcto funcionamiento

del sistema tras la migración, así como analizar su comportamiento en términos de rendimiento, estabilidad y eficiencia.

Para ello, se diseñaron y ejecutaron distintos conjuntos de pruebas, orientados tanto al análisis del motor de eventos Siddhi y su integración en la aplicación, como a la validación del módulo de comunicación *peer-to-peer* (P2P) implementado posteriormente.

A continuación se describen las distintas pruebas realizadas, que se dividen en tres grupos: las pruebas de rendimiento del motor Siddhi, la comparación de rendimiento entre Siddhi y la aplicación completa, y la evaluación del módulo P2P.

5.2.1. Pruebas de rendimiento del motor Siddhi

El procedimiento experimental se basa en el descrito en [6], adaptado a la nueva implementación móvil. Para el experimento se establecieron siete tipos de *context rules*, y se elaboró un script en Python que generaba de forma aleatoria las *triggering rules* combinando dichas reglas de contexto, evitando contradicciones (por ejemplo, `hora=13:30 AND hora=14:27`). Cada *triggering rule* podía contener entre una y siete *context rules*, obteniéndose un total de 300 reglas.

Asimismo, se generaron 21 contextos de prueba reconocibles por Siddhi mediante otro script en Python. Para cada ejecución experimental, se incorporaron a una aplicación Android con Siddhi integrado todas las *context rules* junto con las n primeras *triggering rules* (variando n de forma progresiva hasta 300).

Cada 20 segundos se enviaba un nuevo contexto, y se medía la **latencia de activación de las recomendaciones**, definida como el tiempo transcurrido entre el momento en que el contexto cambiaba en la aplicación y el instante en que el sistema generaba efectivamente la recomendación correspondiente. Es decir, la latencia considerada en este trabajo incluye tanto el tiempo necesario para que el motor detecte la condición de activación como el tiempo adicional requerido para obtener y seleccionar el ítem recomendado.

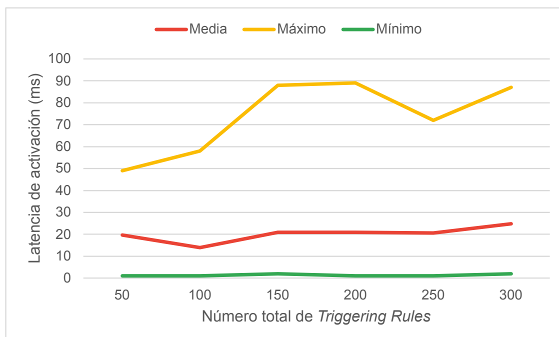
Este criterio difiere del empleado en el trabajo previo que sirve de base a estos experimentos [6], donde se evalúa únicamente la *triggering latency*, entendida como el tiempo desde el cambio en el contexto hasta la detección de la necesidad de disparar una recomendación, sin incluir la fase posterior de generación del ítem recomendado.

Dado que un mismo contexto podía activar múltiples recomendaciones, se registró la latencia de cada una de ellas de forma independiente para cada ejecución experimental. Todas las pruebas se realizaron en un dispositivo *Google Pixel 9 Pro*, garantizando condiciones de ejecución homogéneas. Para más información sobre las características del entorno de evaluación y el hardware utilizado, véase la Sección 5.1.

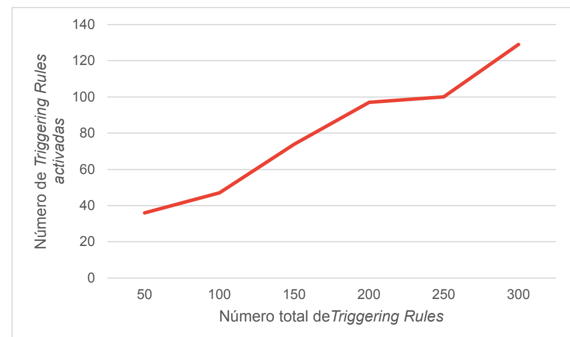
Resultados

En la Figura 5.1 se resumen los resultados obtenidos, que permiten analizar tanto la latencia de activación como el número de reglas disparadas. En la gráfica de la izquierda se representan la latencia media, mínima y máxima en función del número de *triggering rules* definidas. Se observa que la latencia mínima se mantiene prácticamente constante en todas las ejecuciones, mientras que la latencia media permanece relativamente estable conforme aumenta el número de reglas. Sin embargo, la latencia máxima muestra una tendencia creciente, lo cual resulta esperable, ya que a medida que se incrementa el número de reglas a evaluar, es más probable que la última en activarse tarde más en ser detectada.

En la gráfica de la derecha se muestra la relación entre el número total de *triggering rules* definidas y el número de reglas efectivamente activadas. Como cabría esperar, el número de activaciones aumenta con el número de reglas disponibles, confirmando que a mayor complejidad del conjunto de reglas, se incrementa también la cantidad de coincidencias posibles en los contextos evaluados.



(a) Latencia mínima, media y máxima en función del número de *triggering rules*.



(b) Número de reglas activadas según el total de *triggering rules* definidas.

Figura 5.1: Resultados de las pruebas de rendimiento en dispositivo Android con Siddhi.

5.2.2. Comparación de rendimiento Siddhi vs. aplicación completa

Además de las pruebas de rendimiento descritas en la Sección 5.2.1, se llevó a cabo un segundo conjunto de experimentos con el objetivo de analizar el impacto que tiene la integración de Siddhi dentro de la aplicación completa.

La metodología seguida fue la misma que en el experimento anterior: se enviaron contextos de prueba de forma periódica y se midió la latencia de activación de las *triggering rules*. En este caso se empleó un nuevo conjunto de 21 contextos junto con un conjunto de *triggering rules* generadas para la ocasión. Cabe destacar que tanto

en el escenario aislado como en el integrado se utilizaron los mismos contextos y las mismas reglas, garantizando así una comparación justa entre ambos casos.

Todas las pruebas se realizaron utilizando el mismo dispositivo que en el experimento previo, un *Google Pixel 9 Pro*, descrito en la Sección 5.1, lo que asegura la consistencia de las condiciones de ejecución y la validez comparativa de los resultados obtenidos.

Los experimentos se realizaron en dos escenarios diferenciados: (i) la ejecución aislada de Siddhi en una aplicación mínima y (ii) la ejecución de la aplicación completa desarrollada en React Native con el módulo de Siddhi integrado. De este modo fue posible comparar no solo las latencias, sino también el consumo de recursos del dispositivo (uso de CPU, *RSS* y *PSS*) en ambos casos.

Las métricas de uso de recursos se recopilaron mediante la herramienta `adb`, muestreando periódicamente la aplicación en ejecución para obtener datos de CPU y memoria. Para automatizar este proceso se desarrolló un script en `bash`, encargado de invocar `adb`, registrar las métricas y almacenarlas en un archivo `CSV`. Posteriormente, un script complementario en `Python` procesó dichos datos y generó tablas resumen con los valores más representativos de cada experimento.

Ambos scripts se incluyen en el Anexo D. Esta información complementa las medidas de latencia y permite obtener una visión más completa del coste añadido por la capa de la aplicación respecto al motor Siddhi en ejecución aislada.

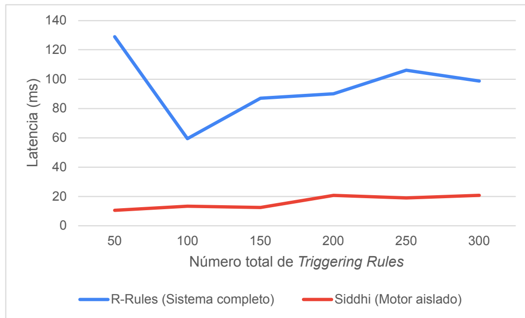
Resultados

En primer lugar, se analizó la latencia de activación de las *triggering rules* en los dos escenarios: Siddhi ejecutado de manera aislada y Siddhi integrado en la aplicación completa. La Figura 5.2 muestra la evolución de la latencia media, máxima y mínima en función del número de reglas de activación.

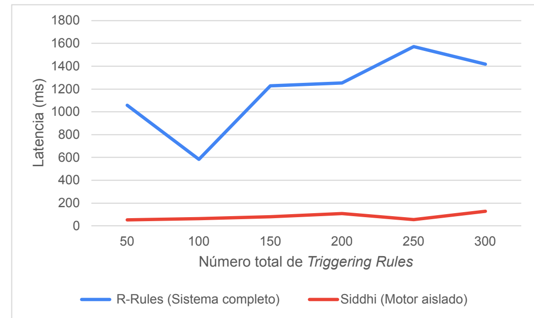
Los resultados muestran que:

- **Latencia media**: mientras que el motor Siddhi aislado mantiene valores relativamente estables (entre ∼10–21 ms), la aplicación completa introduce un incremento significativo. Por ejemplo, con 50 reglas la latencia media pasa de 10,5 ms en Siddhi a 128,86 ms en la aplicación completa, y con 300 reglas de 20,7 ms a 98,72 ms.

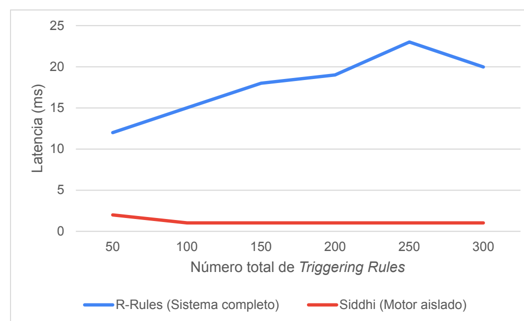
- **Latencia máxima**: la diferencia es aún más marcada, con picos superiores a 1000 ms en la aplicación completa, frente a valores que no superan los 129 ms en Siddhi aislado.



(a) Latencia media



(b) Latencia máxima



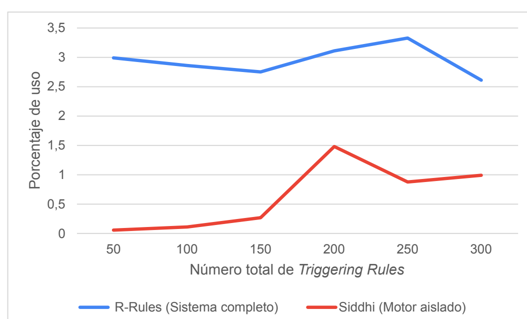
(c) Latencia mínima

Figura 5.2: Comparativa de latencias entre Siddhi aislado y la aplicación completa.

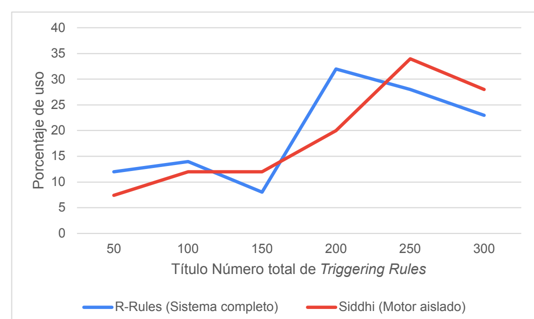
- **Latencia mínima**: se mantiene muy baja en ambos casos, con valores en torno a 1 ms en Siddhi y entre 12 y 23 ms en la aplicación completa.

Estos resultados indican que la sobrecarga introducida por la aplicación completa se refleja principalmente en las medias y en los picos de latencia, mientras que el rendimiento mínimo es similar en ambos escenarios.

En segundo lugar, se analizó el consumo de CPU en ambos escenarios. La Figura 5.3 recoge la evolución del uso medio y máximo de CPU en función del número de reglas de activación.



(a) CPU media



(b) CPU máxima

Figura 5.3: Consumo de CPU en Siddhi aislado y en la aplicación completa.

Los resultados muestran que:

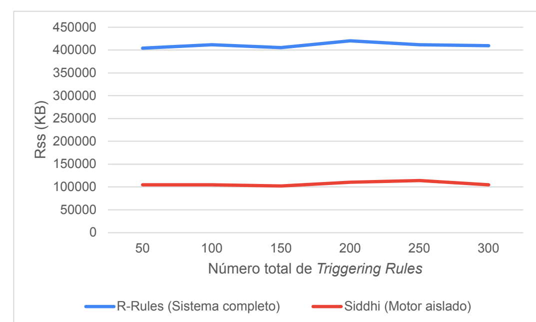
– **CPU media**: el motor Siddhi aislado presenta valores muy bajos (inferiores al 1 % incluso con 300 reglas), mientras que la aplicación completa alcanza entre un 2,6 % y un 3,3 %, con una ligera variación según el número de reglas.

– **CPU máxima**: la diferencia es más acentuada, con picos de hasta 32 % en la aplicación completa, frente a un máximo de 34 % en Siddhi, aunque en la mayoría de los casos los valores del motor aislado se mantienen por debajo.

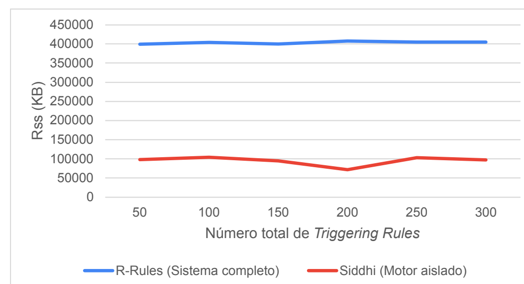
En conjunto, el análisis refleja que la integración en la aplicación completa implica un incremento sostenido en el consumo medio de CPU, mientras que en los valores máximos la diferencia no siempre es tan marcada, dependiendo del número de reglas.

Finalmente, se analizó el consumo de memoria, distinguiendo entre *Resident Set Size* (RSS) y *Proportional Set Size* (PSS). Las Figuras 5.4 y 5.5 muestran la evolución de los valores medio, máximo y mínimo en función del número de reglas.

(a) RSS medio



(b) RSS máximo

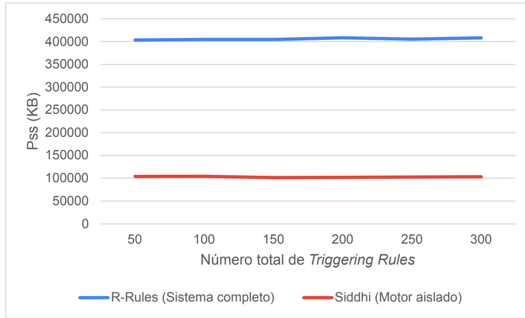


(c) RSS mínimo

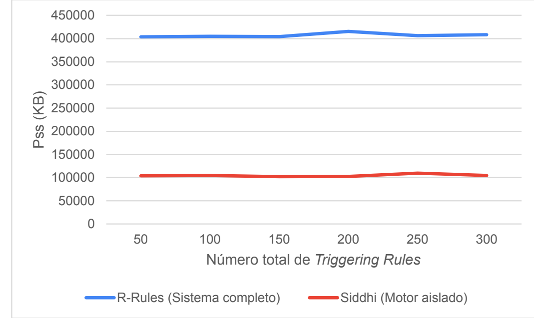
Figura 5.4: Consumo de memoria RSS en Siddhi aislado y en la aplicación completa.

Los resultados muestran que:

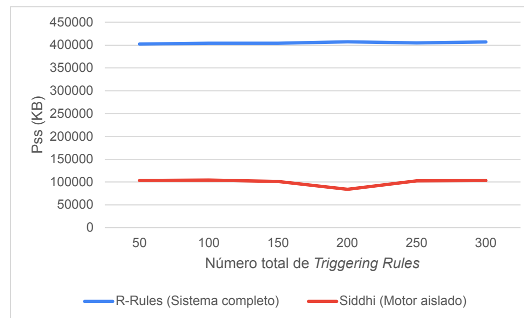
– **RSS**: los valores se sitúan en torno a los 400 MB en la aplicación completa, frente a unos 100 MB en Siddhi aislado. Las variaciones entre mínimo, medio y máximo son reducidas, lo que indica un uso relativamente estable de la memoria en ambos escenarios.



(a) PSS medio



(b) PSS máximo



(c) PSS mínimo

Figura 5.5: Consumo de memoria PSS en Siddhi aislado y en la aplicación completa.

- **PSS**: se observan cifras prácticamente idénticas a las de RSS, tanto en la aplicación completa como en Siddhi. Esto refleja que la diferencia entre RSS y PSS es mínima en este caso.

El análisis de memoria confirma que la integración de Siddhi en la aplicación completa implica un consumo de RAM aproximadamente cuatro veces superior al de la ejecución aislada, aunque la estabilidad se mantiene y las diferencias entre RSS y PSS resultan poco significativas.

El análisis conjunto de las métricas de latencia, CPU y memoria permite extraer varias conclusiones relevantes. En primer lugar, la integración de Siddhi dentro de la aplicación completa introduce una sobrecarga significativa en la **latencia**, especialmente en los valores medios y máximos, mientras que la latencia mínima apenas se ve afectada. Este comportamiento indica que la aplicación añade retrasos adicionales en el procesamiento bajo carga, pero mantiene un rendimiento aceptable en los mejores casos.

En cuanto al **consumo de CPU**, el motor Siddhi aislado muestra valores muy reducidos, mientras que la aplicación completa requiere entre un 2 % y un 3 % de uso sostenido, con picos que alcanzan valores más elevados en determinados escenarios. Esto refleja que la integración incrementa la carga de cómputo, aunque dentro de márgenes razonables para sistemas modernos.

Respecto al **uso de memoria**, la aplicación completa demanda alrededor de 400 MB de RAM frente a los aproximadamente 100 MB de la ejecución aislada. No obstante, tanto RSS como PSS presentan un comportamiento estable y con diferencias mínimas entre sí, lo que confirma que la gestión de memoria es consistente.

En conjunto, los resultados muestran que la aplicación completa supone un coste en términos de latencia y recursos (CPU y RAM) respecto al motor Siddhi aislado, pero mantiene una **estabilidad operativa** y un consumo predecible. Por tanto, la integración es viable desde el punto de vista de rendimiento, siempre que se asuma la sobrecarga inherente al entorno más complejo.

5.2.3. Evaluación del sistema P2P

El propósito de esta sección es analizar el rendimiento y la eficiencia del sistema tras la incorporación del modelo de comunicación *peer-to-peer* (P2P) en *R-Rules*. Para ello, se han diseñado diversos experimentos orientados a evaluar el impacto del intercambio directo de información entre dispositivos. En particular, se estudian métricas relacionadas con la latencia, el volumen de datos transmitidos y el comportamiento del sistema ante escenarios de reconexión.

Escenario de uso

Los experimentos se realizaron utilizando dos dispositivos móviles Android —un Xiaomi Redmi Note 12 y un Vivo Y22s— descritos previamente en la Sección 5.1. Ambos dispositivos ejecutaban la misma versión de la aplicación. El entorno de pruebas se mantuvo controlado: los dispositivos permanecieron a una distancia máxima de dos metros, sin interferencias externas ni otras conexiones activas.

Durante las pruebas, uno de los dispositivos asumió el rol de *emisor* y el otro el de *receptor*. El emisor transmitía una colección de items (actividades) valoradas, que representan las recomendaciones generadas por el sistema. Cada intercambio se efectuó de manera directa, sin intermediarios.

La latencia medida corresponde al **tiempo total de envío y procesamiento del paquete JSON**, es decir, el intervalo comprendido entre el inicio del envío y la confirmación de la finalización del proceso por parte del receptor (registrado desde el dispositivo receptor). No se incluye la latencia de retorno, dado que el protocolo no requiere confirmación de recepción.

Objetivos experimentales

El conjunto de experimentos tiene como propósito analizar los siguientes aspectos del sistema:

1. **Latencia de transmisión:** evaluar el tiempo total requerido para completar el envío de un conjunto de actividades entre ambos dispositivos.

2. **Impacto del volumen de datos:** estudiar cómo varía la latencia en función del número de actividades transmitidas.

3. **Comportamiento ante reconexiones:** analizar la capacidad del sistema para detectar una desconexión, restablecer la conexión y continuar el intercambio de información sin pérdida de datos.

Diseño de los experimentos

Cada experimento se llevó a cabo utilizando distintos tamaños de bloque de actividades (5, 10, 50, 100 y 200 elementos), con el objetivo de evaluar el impacto del volumen de datos en la latencia total de transmisión. Dado que los envíos se realizan en forma de bloques JSON completos, la latencia registrada corresponde al tiempo total de envío y procesamiento de cada bloque, sin necesidad de calcular métricas promedio o extremas para los elementos individuales.

Los experimentos se organizaron en dos grupos principales:

– **Experimento 1: Escalabilidad del volumen de datos.** Se realizaron envíos con distintos tamaños de conjunto (10, 50, 100 y 200 actividades), con el objetivo de analizar cómo el incremento del volumen de datos influye en la latencia total y en la estabilidad de la conexión.

– **Experimento 2: Reconexión y persistencia.** Durante la transmisión de un bloque de datos, se forzó la desconexión temporal de uno de los dispositivos. Posteriormente, se evaluó la capacidad del sistema para reanudar la comunicación y completar el envío, analizando el tiempo adicional requerido y la correcta entrega de todas las actividades.

Tras la ejecución de los experimentos descritos en la sección anterior, se obtuvieron los datos de latencia correspondientes a los distintos escenarios de prueba. Estos resultados permiten analizar tanto el comportamiento del sistema frente a diferentes volúmenes de datos como su capacidad de recuperación ante situaciones de desconexión.

Latencia de transmisión

Se evaluó la latencia de transmisión de bloques JSON que contenían diferentes cantidades de actividades (5, 10, 50, 100 y 200 elementos), con el objetivo de analizar cómo el volumen de datos influye en el tiempo total de envío y procesamiento. Cada

bloque se transmitió de manera completa desde el dispositivo emisor al receptor, registrando la latencia total de la operación.

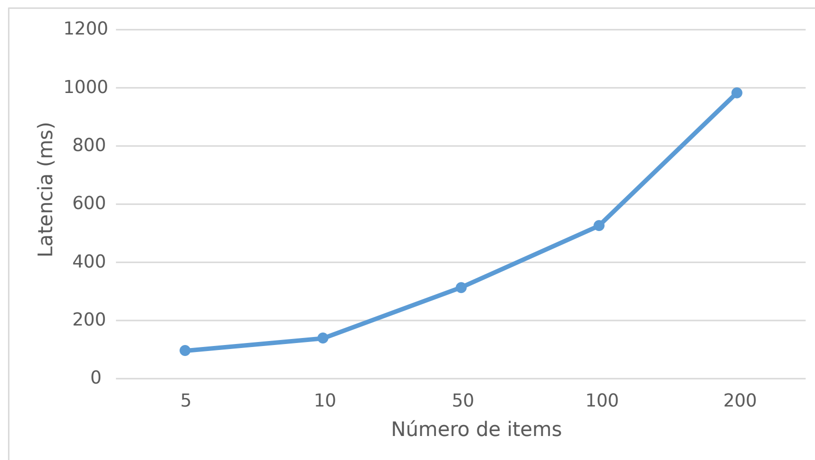


Figura 5.6: Latencia total de transmisión en función del número de actividades enviadas en bloque JSON.

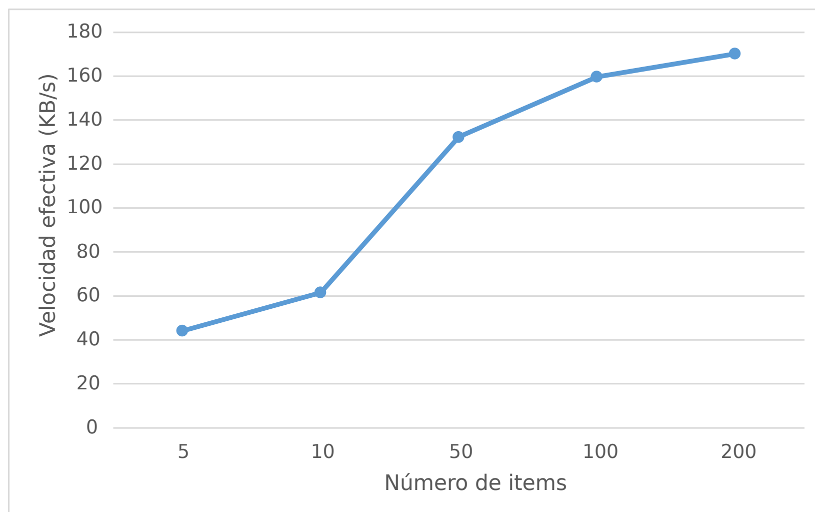


Figura 5.7: Evolución de la velocidad efectiva en función del tamaño del número de actividades enviadas en bloque JSON.

Como se observa en la Figura 5.6, la latencia total aumenta de forma progresiva a medida que crece el tamaño del bloque de actividades. Este incremento mantiene un comportamiento prácticamente lineal en todo el rango analizado, sin que se observe una degradación significativa incluso a partir de los 100 elementos.

Por otro lado, la Figura 5.7 muestra la velocidad efectiva de transmisión en función del tamaño del bloque. Se aprecia que, aunque la latencia absoluta crece, la velocidad efectiva aumenta de forma notable, pasando de 44 KB/s para bloques pequeños a más de 170 KB/s en los mayores. Este comportamiento indica que el sistema gestiona de manera más eficiente los bloques grandes, ya que el coste fijo asociado a la serialización

y al envío de los datos se amortiza progresivamente conforme el volumen de información crece.

En conjunto, los resultados sugieren que el envío agrupado de actividades resulta más eficiente en términos de rendimiento global, siempre que el tamaño de los bloques no comprometa la capacidad de procesamiento del dispositivo receptor.

Comportamiento ante reconexiones

Para evaluar la robustez del sistema, se simuló la pérdida temporal de conexión durante la transmisión de un bloque de actividades. Tras la desconexión, el sistema detectó correctamente la pérdida de enlace e inició el proceso de reconexión automática.

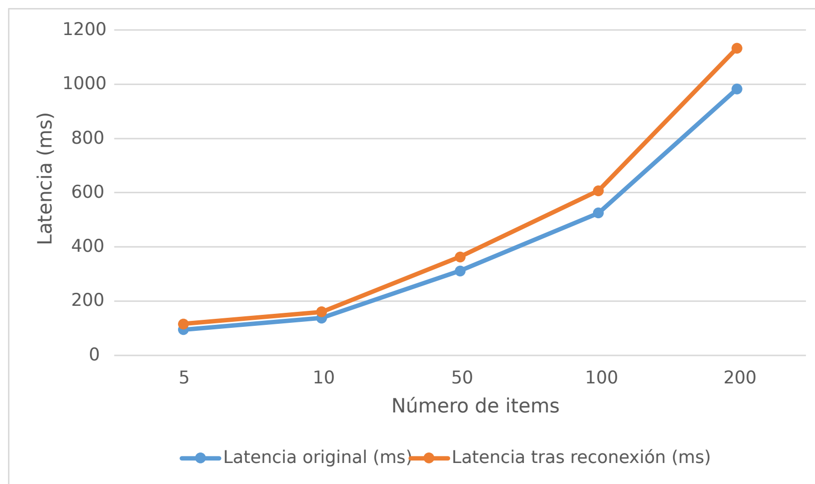


Figura 5.8: Comparativa de la latencia original y tras reconexión para diferentes tamaños de bloque de actividades.

Como se observa en la Figura 5.8, el sistema demuestra una alta tolerancia a fallos. En todos los casos la reconexión se completó con éxito y la entrega de datos fue íntegra, manteniendo un incremento de latencia moderado tras la recuperación. Además, el impacto relativo de la reconexión disminuye a medida que crece el tamaño del bloque, pasando del 22 % para transmisiones pequeñas a aproximadamente un 15 % para las mayores. Este comportamiento indica que el mecanismo de comunicación amortiza eficazmente el coste de la reconexión en operaciones más voluminosas, garantizando estabilidad y continuidad del servicio incluso ante interrupciones temporales.

Resumen de resultados

En conjunto, los resultados obtenidos permiten extraer las siguientes conclusiones principales:

– La latencia total de transmisión aumenta de forma progresiva con el tamaño del

bloque de actividades, reflejando el tiempo requerido para enviar y procesar los datos en el dispositivo receptor.

- El sistema mantiene un comportamiento estable incluso con bloques de gran tamaño (hasta 200 actividades), sin pérdidas de conexión ni errores de transmisión.

- Las reconexiones temporales introducen un retraso adicional moderado (aproximadamente 15–20 %), pero no afectan la integridad ni la entrega completa de los bloques de datos.

Estos resultados confirman que la integración del módulo *P2P* en *R-Rules* garantiza un rendimiento adecuado para el intercambio directo de recomendaciones entre usuarios, ofreciendo eficiencia, fiabilidad y robustez ante interrupciones de la conexión.

5.3. Evaluación de PASEO

Esta sección presenta la evaluación experimental del sistema *PASEO*, con el objetivo de validar el rendimiento y la viabilidad de la arquitectura propuesta. Mientras que en el Capítulo 4 se describió el diseño de la implementación local y la integración del módulo P2P, en este apartado se exponen los resultados obtenidos a partir de pruebas reales realizadas en los dispositivos descritos en la Sección 5.1.

El propósito principal de la evaluación es comparar el comportamiento de la aplicación bajo dos configuraciones arquitectónicas:

- **Arquitectura cliente-servidor original:** modelo centralizado en el que la lógica de procesamiento y el intercambio de datos dependen de un servidor remoto.

- **Arquitectura local propuesta:** nueva versión que traslada el procesamiento al dispositivo del usuario, apoyándose en un módulo **P2P** para la comunicación directa entre nodos y la sincronización de información.

A través de esta comparación se busca cuantificar el impacto de la descentralización en términos de latencia, uso de memoria y estabilidad, así como validar el correcto funcionamiento del módulo P2P dentro de la solución local. Los resultados permiten determinar hasta qué punto la arquitectura propuesta mejora la eficiencia y la autonomía del sistema frente al modelo tradicional.

5.3.1. Comparativa entre arquitecturas cliente-servidor y local

Se realizaron mediciones comparativas sobre las dos versiones del sistema descritas anteriormente, con el fin de analizar el impacto de la arquitectura en el rendimiento y el consumo de recursos. Las pruebas se llevaron a cabo en los mismos dispositivos definidos en la Sección 5.1, ejecutando idénticos escenarios de uso para garantizar la comparabilidad de los resultados.

En ambos casos se evaluaron métricas clave orientadas a cuantificar el rendimiento de las arquitecturas **cliente-servidor** y **local en dispositivo**:

- **Latencia**: tiempo de respuesta medio en operaciones de consulta y procesamiento de datos.

- **Uso de CPU**: porcentaje de carga del procesador durante la ejecución de tareas intensivas.

- **Consumo de RAM**: memoria ocupada por la aplicación en distintos escenarios de interacción.

El análisis de estos indicadores permite determinar hasta qué punto la ejecución local mejora la autonomía y la eficiencia del sistema, así como identificar los posibles costes computacionales asociados a la descentralización.

Para la evaluación de la latencia se diseñó un procedimiento homogéneo aplicable a ambas versiones (local y cliente-servidor) en los dos dispositivos:

1. Se realizaron un total de 100 mediciones por arquitectura, organizadas en 10 grupos de 10 ejecuciones consecutivas.

2. Cada medición recoge el tiempo total de respuesta, incluyendo tanto la ejecución de la consulta como la entrega de resultados a la aplicación.

3. Este enfoque permitió calcular valores promedio, mínimos y máximos, reduciendo la influencia de fluctuaciones puntuales del sistema o de la red.

El código empleado para la automatización de estas pruebas se incluye en el Anexo *"Test de latencia implementados"* (ver Anexo C).

El análisis del consumo de recursos se llevó a cabo en ambos dispositivos mediante un procedimiento que combina herramientas nativas de Android con scripts personalizados.

El análisis del consumo de recursos se llevó a cabo directamente en los dispositivos Android utilizados para las pruebas, sin recurrir a emuladores ni a mediciones en la máquina servidora. Para ello, se empleó un script en Bash que, a través de `adb`, recopila periódicamente la información asociada al proceso de la aplicación en ejecución.

- **CPU %:** porcentaje de uso del procesador.
- **RSS (Resident Set Size):** memoria residente en RAM ocupada por el proceso.
- **PSS (Proportional Set Size):** memoria proporcionalmente atribuida al proceso.

Los datos de cada dispositivo se almacenaron en archivos CSV para su posterior análisis. Posteriormente, un script en Python procesó los registros y calculó los valores mínimo, máximo y promedio de cada métrica.

Tanto el script de recogida como el de análisis se incluyen en el Anexo *"Medición de métricas"* (ver Anexo D).

Pruebas de latencia

Para cada arquitectura y dispositivo se realizaron 100 mediciones de latencia, organizadas en 10 grupos de 10 ejecuciones consecutivas. Cada grupo se configuró con un conjunto distinto de parámetros de latitud, longitud y distancia máxima, lo que hace que las consultas ejecutadas sobre el sistema fueran diferentes entre grupos. Con el fin de facilitar el análisis, los resultados se presentan en tablas que recogen, para cada grupo, los valores de latencia mínima, máxima y media (expresados en milisegundos). Esta configuración permite evaluar cómo varía la latencia en función de la localización y el radio de búsqueda empleados en las recomendaciones.

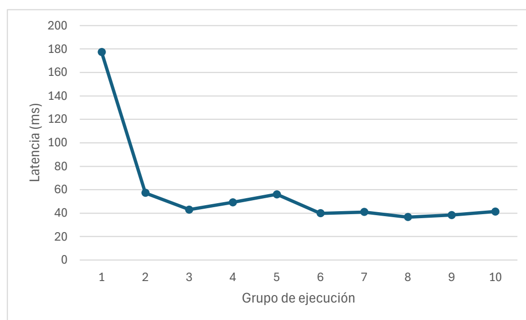
Los resultados obtenidos en el Xiaomi Redmi Note 10S se resumen en las Tablas 5.1 y 5.2. Como complemento, en las Figuras 5.9 y 5.10 se representan gráficamente los valores medios de latencia para cada grupo, donde los datos están organizados en 10 grupos, cada uno con 10 ejecuciones. En estas figuras, los círculos más gruesos indican los valores medios de cada grupo, mientras que las barras de error representan los intervalos de confianza al 95 % calculados a partir de la desviación estándar de las ejecuciones correspondientes. Además, se incluyen etiquetas que muestran la desviación estándar de cada punto. Esta representación permite observar con claridad tanto la tendencia central como la variabilidad de los resultados, facilitando la comparación entre las arquitecturas evaluadas.

Tabla 5.1: Resultados de latencia en el Xiaomi Redmi Note 10S (ejecución en local).

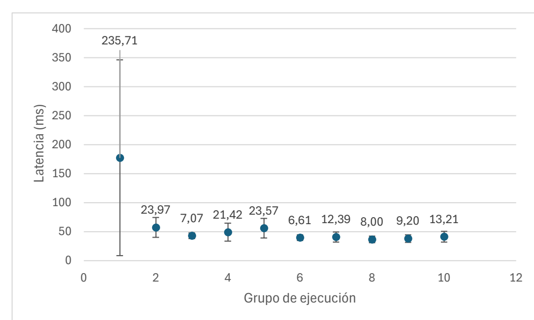
Grupo	**Latencia mínima (ms)**	**Latencia máxima (ms)**	**Latencia media (ms)**
1	49	825	177.4
2	33	111	57.3
3	31	58	43.0
4	30	97	49.3
5	30	87	56.0
6	31	50	39.9
7	30	71	41.0
8	31	56	36.7
9	30	60	38.4
10	30	73	41.4

Tabla 5.2: Resultados de latencia en el Xiaomi Redmi Note 10S (arquitectura cliente-servidor).

Grupo	**Latencia mínima (ms)**	**Latencia máxima (ms)**	**Latencia media (ms)**
1	2424	4948	2801.7
2	2370	2730	2492.7
3	2353	2541	2456.1
4	2301	2689	2445.0
5	2345	2581	2435.2
6	2376	3230	2561.3
7	2311	2473	2418.4
8	2298	2724	2447.7
9	2288	2688	2462.7
10	2398	2724	2501.0

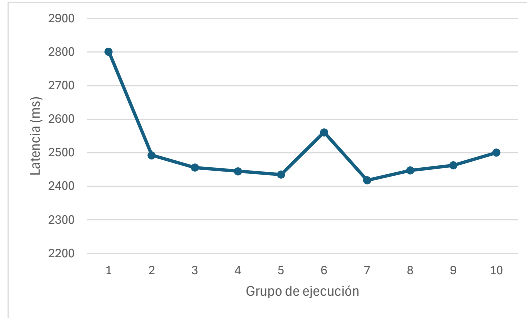


(a) Latencia media en ejecución local

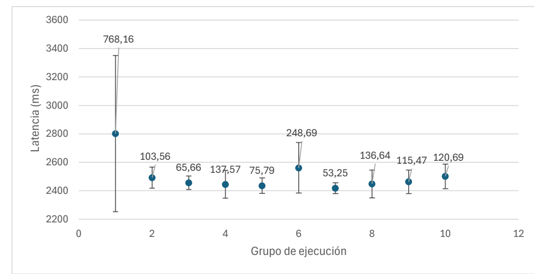


(b) Dispersión de latencias en ejecución local

Figura 5.9: Resultados de latencia en el Xiaomi Redmi Note 10S para la arquitectura local.



(a) Latencia media en cliente-servidor



(b) Dispersión de latencias en cliente-servidor

Figura 5.10: Resultados de latencia en el Xiaomi Redmi Note 10S para la arquitectura cliente-servidor.

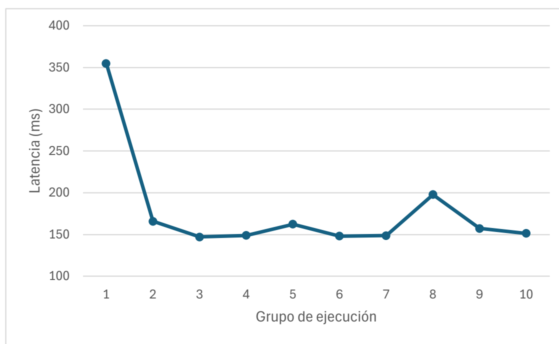
Los resultados obtenidos en el Vivo Y36 se resumen en las Tablas 5.3 y 5.4. Como complemento, en las Figuras 5.11a y 5.12a se representan gráficamente los valores medios de latencia para cada grupo, donde los datos están organizados en 10 grupos, cada uno con 10 ejecuciones. En estas figuras, los círculos más gruesos indican los valores medios de cada grupo, mientras que las barras de error representan los intervalos de confianza al 95 % calculados a partir de la desviación estándar de las ejecuciones correspondientes. Además, se incluyen etiquetas que muestran la desviación estándar de cada punto. Esta representación permite observar con claridad tanto la tendencia central como la variabilidad de los resultados, facilitando la comparación entre las arquitecturas analizadas en un dispositivo con mayores recursos de hardware.

Tabla 5.3: Resultados de latencia en el Vivo Y36 (ejecución en local).

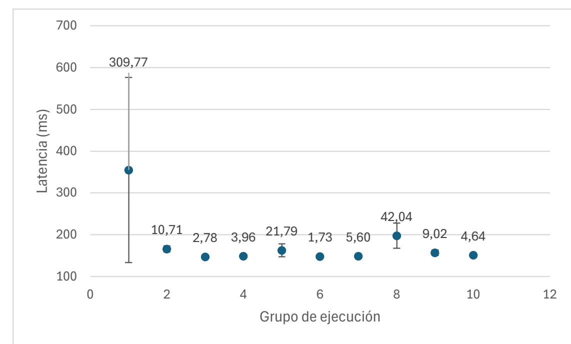
Grupo	**Latencia mínima (ms)**	**Latencia máxima (ms)**	**Latencia media (ms)**
1	189	1217	354.8
2	154	181	165.9
3	141	151	147.2
4	144	156	148.9
5	142	203	162.4
6	145	151	148.1
7	140	160	148.7
8	162	310	197.9
9	148	177	157.2
10	145	161	151.3

Tabla 5.4: Resultados de latencia en el Vivo Y36 (arquitectura cliente-servidor).

Grupo	**Latencia mínima (ms)**	**Latencia máxima (ms)**	**Latencia media (ms)**
1	2277	3335	2590.5
2	2337	2499	2426.0
3	2342	3121	2494.7
4	2335	2536	2433.0
5	2338	3039	2541.4
6	2313	2491	2435.7
7	2222	2824	2507.5
8	2287	2639	2432.2
9	2327	3429	2531.6
10	2338	2768	2486.7

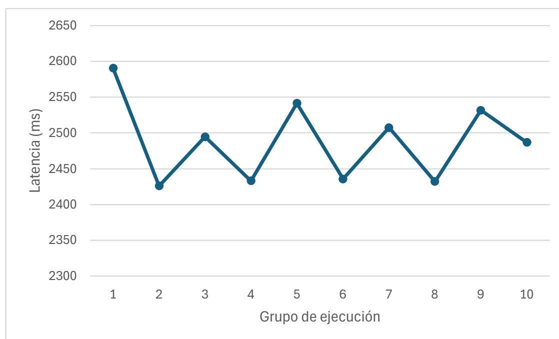


(a) Latencia media por grupo (local).

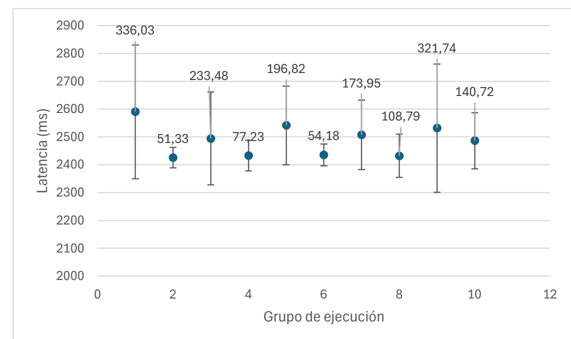


(b) Dispersión de la latencia (local).

Figura 5.11: Resultados de latencia en arquitectura local para el dispositivo Vivo Y36.



(a) Latencia media por grupo (cliente-servidor).



(b) Dispersión de la latencia (cliente-servidor).

Figura 5.12: Resultados de latencia en arquitectura cliente-servidor para el dispositivo Vivo Y36.

Análisis de resultados

Los resultados obtenidos ponen de manifiesto una diferencia muy significativa entre las dos arquitecturas. En la versión local, las consultas se ejecutan en tiempos muy

reducidos, con medias comprendidas entre 36 y 57 ms en el Xiaomi Redmi Note 10S y entre 147 y 197 ms en el Vivo Y36 (una vez superado el primer grupo de pruebas). En la versión cliente-servidor, los tiempos de respuesta son varios órdenes de magnitud superiores, situándose de forma consistente entre 2,4 y 2,8 segundos en ambos dispositivos. Esto se debe a la dependencia de la red y al tiempo de comunicación con el servidor, que introduce una latencia estructural imposible de eliminar.

En ambos dispositivos se observa un comportamiento anómalo en el Grupo 1 de la versión local, con latencias máximas y medias muy superiores al resto. En el Xiaomi Redmi Note 10S se observa una media de 177,4 ms frente a valores cercanos a los 40 ms en los grupos siguientes. En el Vivo Y36 se observa una media de 354,8 ms, reduciéndose después de forma estable por debajo de los 200 ms. Este efecto puede explicarse por la fase de *warm-up* o inicialización: la primera ejecución requiere la carga de bibliotecas, el arranque del motor de persistencia (Room) y la creación de cachés internas, lo que impacta temporalmente en el rendimiento. Una vez completado este proceso, los tiempos se estabilizan.

Los resultados muestran además un contraste inesperado entre ambos terminales. En la solución local, el Xiaomi Redmi Note 10S ofrece tiempos de respuesta significativamente mejores, con latencias medias en torno a 40–55 ms frente a los 150–190 ms del Vivo Y36 (una diferencia de aproximadamente 3–4 veces menos). En la solución cliente-servidor, las diferencias se reducen prácticamente a cero, con latencias medias muy similares entre ambos dispositivos (aprox. 2,4–2,6 s).

Este comportamiento indica que la ejecución local está mucho más influida por la arquitectura interna del procesador y la optimización del sistema operativo, mientras que en el enfoque cliente-servidor el factor determinante es la latencia de red.

Pruebas de CPU y RAM

Además de la latencia, se evaluó el consumo de recursos internos del dispositivo, en particular el uso de CPU y el consumo de memoria RAM. Estas métricas resultan críticas para valorar la eficiencia de la aplicación en cada arquitectura, ya que determinan no solo la fluidez de la experiencia de usuario, sino también la capacidad de ejecución en segundo plano y el impacto en la autonomía de la batería.

Para ello, se aplicó la siguiente metodología: se ejecutaron ambas versiones de la aplicación en cada dispositivo por separado, durante un tiempo de dos minutos por prueba. En todas las ejecuciones se realizaron el mismo número de acciones, incluyendo un total de 12 búsquedas de términos y la adición de 10 resultados a la pestaña de favoritos, garantizando así la comparabilidad entre arquitecturas y dispositivos.

Cabe destacar que, antes de cada prueba, toda la información relativa a la aplicación

fue eliminada, con el fin de evitar desajustes derivados de la caché y poder evaluar de manera precisa la carga inicial de la solución local en el dispositivo, reflejando el comportamiento real que experimentaría un usuario al iniciar la aplicación por primera vez.

Para cada arquitectura (local y cliente-servidor) se recopilaron de manera periódica las siguientes métricas durante la ejecución de operaciones representativas:

– **CPU %**: porcentaje de utilización del procesador atribuida a la aplicación.

– **RSS (Resident Set Size)**: cantidad de memoria residente ocupada en la RAM por el proceso.

– **PSS (Proportional Set Size)**: memoria proporcionalmente asignada al proceso, teniendo en cuenta bibliotecas compartidas.

Las mediciones se realizaron en los dos dispositivos de prueba (Xiaomi Redmi Note 10S y Vivo Y36), con el fin de obtener una perspectiva más completa:

– En el **Xiaomi Redmi Note 10S** se buscó observar cómo se comporta la aplicación en un terminal de gama media con especificaciones equilibradas.

– En el **Vivo Y36**, con mayor cantidad de memoria y un procesador distinto, se buscó contrastar si el incremento de recursos repercute directamente en una reducción del consumo relativo o en una mejor distribución de la carga.

Los resultados obtenidos se presentan en forma de tablas que recogen los valores mínimo, máximo y promedio para cada métrica y dispositivo. Las Tablas 5.5 y 5.6 muestran estos resultados para el Xiaomi Redmi Note 10S y el Vivo Y36, respectivamente.

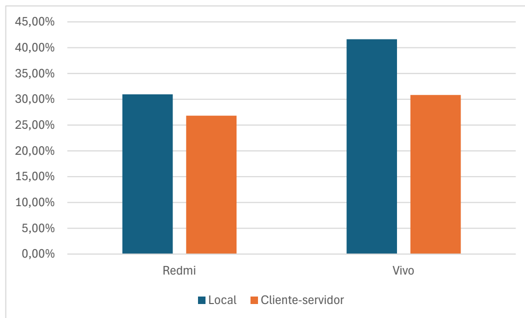
Adicionalmente, en las Figuras 5.13a, 5.13b y 5.13c se comparan las medias de cada métrica entre los dispositivos y arquitecturas estudiadas.

Tabla 5.5: Uso de CPU y memoria en el Xiaomi Redmi Note 10S.

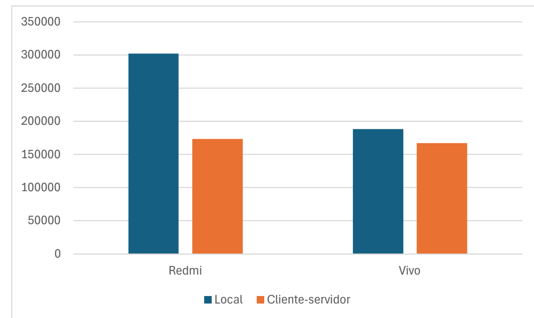
Arquitectura	Métrica	Media	Mínimo	Máximo
Local	CPU %	30.95 %	0.00 %	106.00 %
Local	RSS KB	305,420	238,290	382,774
Local	PSS KB	302,025	236,353	381,759
Cliente-servidor	CPU %	26.79 %	0.00 %	82.10 %
Cliente-servidor	RSS KB	189,451	132,604	278,699
Cliente-servidor	PSS KB	188,284	140,381	277,771

Tabla 5.6: Uso de CPU y memoria en el Vivo Y36.

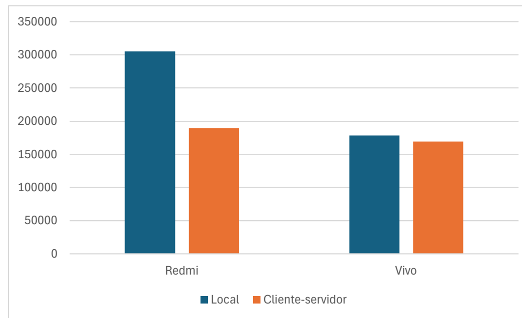
Arquitectura	Métrica	Media	Mínimo	Máximo
Local	CPU %	41.62 %	0.00 %	163.00 %
Local	RSS KB	178,513	145,049	230,471
Local	PSS KB	173,280	145,297	195,740
Cliente-servidor	CPU %	30.84 %	0.00 %	87.00 %
Cliente-servidor	RSS KB	169,279	129,555	193,195
Cliente-servidor	PSS KB	167,005	135,845	192,646



(a) Media de uso de CPU.



(b) Media de PSS.



(c) Media de RSS.

Figura 5.13: Comparación de las medias de CPU, PSS y RSS entre dispositivos y arquitecturas.

Análisis de resultados

Los resultados muestran diferencias interesantes tanto entre las arquitecturas como entre los dispositivos. En la solución local, la aplicación presenta un consumo de CPU considerablemente mayor, con medias de 30,95 % en el Xiaomi Redmi Note 10S y 41,62 % en el Vivo Y36, lo que indica que, al ejecutar toda la lógica de acceso y gestión de datos en el dispositivo, el procesador se ve más cargado durante las operaciones. En la solución cliente-servidor, el consumo de CPU se reduce notablemente en ambos terminales (26,79 % en Xiaomi y 30,84 % en Vivo), ya que gran parte del procesamiento se realiza en el servidor remoto, mientras que el dispositivo solo gestiona la comunicación y el renderizado de resultados.

En cuanto a la memoria RAM, en la versión local el Xiaomi utiliza más memoria (305 MB RSS) que el Vivo (178 MB RSS), a pesar de que este último dispone de más memoria total. Esto sugiere que la implementación local en el Xiaomi está más optimizada o que su motor de persistencia gestiona la caché de manera más agresiva. En la versión cliente-servidor, el consumo de RAM disminuye en ambos dispositivos, reflejando que los datos no se almacenan completamente en el dispositivo, sino que se reciben y procesan según necesidad.

Si comparamos ambos dispositivos, se observa que en el Xiaomi la CPU alcanza picos más bajos que en el Vivo (106 % vs. 163 %). Aunque estos valores puedan parecer superiores al 100 %, en Android la métrica de CPU se calcula como la suma de la carga de todos los núcleos disponibles, por lo que es posible superar el 100 % en dispositivos con múltiples núcleos. En este caso, el valor refleja que la carga se distribuye entre varios núcleos, lo que permite mantener la ejecución local fluida. Por otra parte, la RAM utilizada es mayor en el Xiaomi, lo que indica un balance entre memoria y procesador que favorece la ejecución local y explica la menor latencia observada. En el Vivo, aunque dispone de más RAM y un procesador más moderno, la CPU se carga más en la versión local y la memoria utilizada es menor, lo que sugiere que este dispositivo realiza más trabajo por CPU para gestionar las mismas operaciones, posiblemente debido a la optimización del sistema operativo (*Funtouch OS*) o al comportamiento del motor de persistencia en este hardware.

Conclusiones comparativas entre soluciones

El análisis realizado evidencia diferencias significativas entre la arquitectura local y la arquitectura cliente-servidor, tanto a nivel de rendimiento como de experiencia de usuario:

Latencia La arquitectura local ofrece tiempos de respuesta prácticamente instantáneos una vez superada la fase de *warm-up*, con medias en torno a 40–50 ms en el Xiaomi y 150–200 ms en el Vivo. En cambio, la arquitectura cliente-servidor presenta latencias constantes de 2,4–2,8 s, determinadas por la comunicación con el servidor. Esto hace que la ejecución local sea claramente superior en términos de rapidez, especialmente en operaciones frecuentes o sensibles al tiempo de espera.

Consumo de recursos La solución local implica un mayor uso de CPU y, en algunos casos, de RAM, ya que todo el procesamiento y almacenamiento de datos se realiza en el dispositivo. Por el contrario, la arquitectura cliente-servidor reduce significativamente la carga del dispositivo, al delegar gran parte del procesamiento en el servidor, aunque

a costa de la latencia elevada.

Comparación entre dispositivos Los resultados muestran que el rendimiento local está altamente condicionado por la arquitectura interna y la optimización del sistema operativo. El Xiaomi Redmi Note 10S logra menores latencias y una utilización de CPU más equilibrada, mientras que el Vivo Y36 presenta un mayor consumo de CPU y latencias locales más altas, pese a contar con más RAM y un procesador más reciente. En la versión cliente-servidor, las diferencias entre dispositivos son prácticamente nulas, ya que la latencia de red domina el rendimiento.

La elección entre ambas arquitecturas depende del equilibrio entre rendimiento y eficiencia de recursos. La solución local es recomendable cuando la velocidad y la inmediatez de la respuesta son prioritarias, mientras que la arquitectura cliente-servidor es más adecuada en escenarios donde la centralización de datos y la reducción del consumo del dispositivo son críticas. Desde la perspectiva de la experiencia de usuario, la latencia perceptible podría tener un impacto directo en la satisfacción y la eficiencia del uso de la aplicación.

Además, una ventaja fundamental de la versión local es que puede operar sin conexión a internet, lo que permite al usuario acceder a la información incluso en entornos donde la conectividad es limitada o inexistente, como puede ocurrir en algunas zonas turísticas. Aunque esta modalidad implica la pérdida de sincronización inmediata de los datos, resulta muy beneficiosa en este escenario, ya que garantiza acceso continuo a datos esenciales, mejorando la autonomía y la experiencia de los usuarios durante su desplazamiento.

5.3.2. Evaluación del sistema P2P

La arquitectura *peer-to-peer* (P2P) implementada en PASEO permite que los dispositivos intercambien información contextual y actividades detectadas localmente sin depender de un servidor central. Este enfoque descentralizado posibilita que cada nodo en la red enriquezca su propio conocimiento con los datos recibidos de otros pares, con el objetivo de aumentar la cobertura y la precisión de las recomendaciones generadas.

El propósito de esta evaluación es analizar el impacto de la comunicación P2P en la calidad del proceso de recomendación, complementando las pruebas de rendimiento realizadas previamente en *R-Rules*. Mientras que en dicho sistema el análisis se centró en la eficiencia de transmisión y la estabilidad de la conexión, en *PASEO* el interés se traslada hacia la influencia que tiene el intercambio de información entre nodos sobre la relevancia y utilidad de las recomendaciones resultantes.

Este cambio de enfoque responde a la distinta naturaleza de ambos sistemas: *R-Rules* sirvió principalmente como banco de pruebas para validar la viabilidad técnica del mecanismo P2P y su integración con el motor de reglas, mientras que *PASEO*, al incorporar además un sistema de recomendación completo, permite evaluar el impacto de la comunicación entre dispositivos en la calidad final de las sugerencias generadas. De este modo, los experimentos realizados en *PASEO* no repiten los de *R-Rules*, sino que los amplían hacia un nivel funcional más alto.

Escenario de uso

Los experimentos se realizaron utilizando los mismos dispositivos móviles empleados en las pruebas de R-Rules: un Xiaomi Redmi Note 12 y un Vivo Y22s, ambos descritos previamente en la Sección 5.1. Los dos terminales ejecutaban la misma versión de la aplicación PASEO, configurada con parámetros de conexión y sincronización equivalentes.

El entorno de pruebas se mantuvo controlado con el fin de minimizar la influencia de factores externos. Los dispositivos se colocaron a una distancia máxima de dos metros, en un espacio sin interferencias ni otros dispositivos cercanos utilizando conexiones de proximidad. No se desactivaron las interfaces inalámbricas, ya que el sistema de comunicación se basa en la API *Nearby Connections* de Android, que emplea tecnologías como Bluetooth y Wi-Fi Direct para el descubrimiento y la transmisión.

Durante los experimentos, ambos dispositivos alternaron entre los roles de *emisor* y *receptor*. El emisor compartía la información contextual y las actividades registradas localmente —equivalentes a los elementos utilizados para generar recomendaciones—, mientras que el receptor integraba los datos recibidos en su propio modelo local de recomendación.

Objetivos experimentales

El propósito de los experimentos fue analizar el impacto del intercambio P2P en la calidad y relevancia de las recomendaciones generadas por el sistema, partiendo de que cada dispositivo representa a un usuario con intereses y contextos diferenciados.

A diferencia de las pruebas realizadas en R-Rules —centradas en la eficiencia de transmisión—, en esta evaluación se busca determinar si la comunicación directa entre usuarios contribuye a mejorar la utilidad real de las recomendaciones, sin comprometer su adecuación al perfil individual de cada participante.

Con este fin, se establecieron los siguientes objetivos específicos:

1. **Precisión de las recomendaciones:** evaluar si, tras el intercambio P2P,

los dispositivos mantienen un alto grado de correspondencia entre las recomendaciones ofrecidas y los intereses particulares del usuario.

2. **Cobertura personalizada:** analizar si la información procedente de otros pares amplía la variedad de recomendaciones relevantes, incorporando nuevas categorías útiles sin introducir resultados irrelevantes.

3. **Control de irrelevancia:** cuantificar el porcentaje de recomendaciones no relacionadas con las preferencias del usuario, con el fin de verificar que la difusión de información no afecta negativamente a la personalización.

Diseño de los experimentos

Para cada uno de los objetivos definidos se diseñó un procedimiento controlado compuesto por tres fases: generación de recomendaciones iniciales, intercambio P2P entre dispositivos y evaluación posterior del cambio producido en la calidad de las recomendaciones.

Cada dispositivo generó un conjunto inicial de recomendaciones basadas en su información local. Tras el intercambio P2P, en el que los nodos compartieron sus actividades y valoraciones, se generó nuevamente el conjunto de recomendaciones. La precisión se midió comparando los resultados obtenidos con las preferencias reales del usuario (modelo de referencia). Para ello se utilizó el *índice de acierto* (P), definido como el porcentaje de recomendaciones relevantes sobre el total de recomendaciones generadas:

$$P = \frac{R_{relevantes}}{R_{totales}} \times 100$$

donde $R_{relevantes}$ representa el número de elementos coincidentes con las actividades valoradas positivamente por el usuario.

Para analizar si el intercambio de información amplía la variedad de recomendaciones sin perder relevancia, se midió el número total de elementos distintos recomendados a cada usuario (N) antes y después del intercambio. Asimismo, se calculó el índice de diversidad (D), que cuantifica la proporción de categorías o tipos de actividades nuevas introducidas tras la comunicación entre pares:

$$D = \frac{C_{nuevas}}{C_{totales}} \times 100$$

Un incremento en N o D sugiere una mejora en la cobertura del sistema, indicando que la información proveniente de otros usuarios aporta variedad útil al conjunto de recomendaciones.

Finalmente, se analizó el impacto del intercambio P2P en la proporción de recomendaciones no pertinentes. El *índice de irrelevancia* (I) se definió como el porcentaje de recomendaciones que no se ajustan al perfil de preferencias del usuario:

$$I = \frac{R_{irrelevantes}}{R_{totales}} \times 100$$

Una disminución de I o su mantenimiento tras el intercambio indica que el proceso de difusión de conocimiento entre pares no degrada la personalización de las recomendaciones.

Para determinar si una recomendación resulta relevante para un usuario, se empleó un enfoque basado en coincidencia de términos. Cada perfil de usuario se asoció con un conjunto de palabras clave que representan sus intereses principales. Un punto de interés se consideró relevante si contenía al menos uno de estos términos en su nombre o descripción.

Formalmente, la relevancia de un punto de interés poi para un usuario u se definió como:

$$rel(u, poi) = \begin{cases} 1, & \text{si } \exists\, t \in T_u \text{ tal que } t \in \text{texto}(poi) \\ 0, & \text{en caso contrario} \end{cases}$$

donde T_u representa el conjunto de términos asociados al perfil del usuario. Esta definición permitió calcular el índice de precisión (P) de forma automática y coherente con los intereses semánticos definidos en el sistema.

Procedimiento experimental

Para la ejecución de las pruebas se utilizaron dos dispositivos móviles, cada uno asociado a un usuario con un perfil de intereses diferente. Para facilitar la descripción, los perfiles se denominaron de la siguiente forma:

- **Ana (perfil familiar):** base de datos inicial compuesta por actividades relacionadas con el ocio infantil y educativo (parques, museos interactivos, zonas recreativas).

- **Carlos (perfil social y cultural):** base de datos inicial con actividades de ocio nocturno y cultural (cines, cafeterías, salas de conciertos).

Cada usuario se asoció a un conjunto de términos representativos de sus intereses principales, que se utilizaron para evaluar la relevancia de las recomendaciones. Los conjuntos definidos fueron:

- **Ana:** *parque*, *museo*, *niños*, *educativo*, *recreativo.*

- **Carlos:** *cine*, *cafetería*, *concierto*, *bar*, *cultural*.

Cada dispositivo generó un primer conjunto de recomendaciones utilizando únicamente su información local. A continuación, se realizó un intercambio P2P en el que ambos compartieron sus historiales de interacción y valoraciones. Una vez completada la sincronización, cada uno recalculó sus recomendaciones incorporando los nuevos datos recibidos.

El procedimiento completo se repitió tres veces, alternando qué dispositivo iniciaba la comunicación, con el objetivo de comprobar la consistencia de los resultados y descartar posibles sesgos debidos al rol de emisor o receptor.

Tras cada ejecución se compararon las recomendaciones obtenidas antes y después del intercambio, aplicando las métricas definidas previamente:

- **Precisión** (P): porcentaje de recomendaciones relevantes.
- **Cobertura** y **diversidad** (N y D): número de elementos distintos y proporción de nuevas categorías sugeridas.
- **Irrelevancia** (I): porcentaje de recomendaciones no relacionadas con el perfil del usuario.

Los valores se registraron para ambos usuarios y se analizaron de forma comparativa para identificar tendencias. Se prestó especial atención a observar si el intercambio de información entre pares mejoraba la utilidad de las recomendaciones sin comprometer su personalización. La relevancia de cada recomendación se determinó aplicando el criterio de coincidencia de términos descrito en la sección anterior.

Para asegurar la reproducibilidad, las condiciones de red, los historiales iniciales y las configuraciones del sistema se mantuvieron constantes en todas las ejecuciones.

Resumen de resultados

Los resultados obtenidos tras la ejecución de los experimentos se presentan en la Tabla 5.7, donde se recogen los valores de precisión, diversidad e irrelevancia calculados para cada usuario antes y después del intercambio P2P de información.

Tabla 5.7: Resultados experimentales del intercambio P2P.

Usuario	**Fase**	**Precisión (P) [%]**	**Diversidad (D) [%]**	**Irrelevancia (I) [%]**
Ana	Antes del intercambio	72	15	12
Ana	Después del intercambio	78	24	11
Carlos	Antes del intercambio	69	10	14
Carlos	Después del intercambio	75	21	12

Los resultados evidencian una tendencia consistente de mejora tras el intercambio P2P para ambos usuarios. En el caso de **Ana**, el índice de precisión aumentó del 72 % al 78 %, indicando que las recomendaciones generadas después de la sincronización se ajustaron mejor a sus intereses de tipo familiar y educativo. Asimismo, la diversidad de las sugerencias se amplió notablemente (del 15 % al 24 %), lo que sugiere que la información procedente del dispositivo de Carlos introdujo nuevas categorías de actividades sin afectar negativamente la coherencia de su perfil.

De forma similar, **Carlos** experimentó un incremento en la precisión, pasando del 69 % al 75 %, acompañado de un aumento considerable en la diversidad (del 10 % al 21 %). Esto refleja que la comunicación entre pares permitió incorporar opciones de ocio que, aunque no estaban presentes inicialmente en su base local, resultaron acordes a sus preferencias culturales y sociales.

Por último, el *índice de irrelevancia* se mantuvo estable o descendió ligeramente en ambos casos (Ana: de 12 % a 11 %; Carlos: de 14 % a 12 %), lo que confirma que el intercambio de información no introdujo ruido significativo ni deterioró la personalización de las recomendaciones. Este resultado era esperable, dado que el sistema no presenta un aumento potencial de irrelevancia: aunque un dispositivo reciba datos o valoraciones que no se ajusten a las preferencias del usuario, dichas recomendaciones son filtradas localmente antes de mostrarse. De este modo, el módulo P2P amplía la base de conocimiento compartido sin comprometer la coherencia ni la pertinencia del conjunto de sugerencias finales.

En conjunto, los resultados respaldan la hipótesis planteada: el intercambio P2P contribuye a mejorar tanto la utilidad como la variedad de las recomendaciones sin comprometer su adecuación al perfil del usuario. Este comportamiento refuerza la viabilidad del enfoque propuesto, sugiriendo que la difusión controlada de conocimiento entre dispositivos puede constituir una estrategia efectiva para enriquecer sistemas de recomendación distribuidos como PASEO.

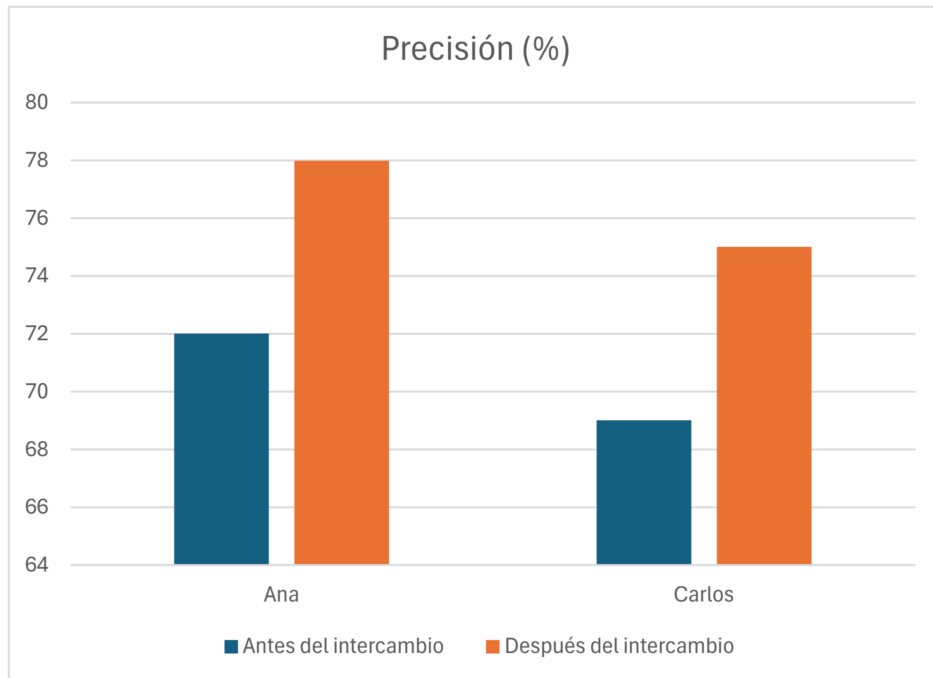


Figura 5.14: Comparación del índice de precisión antes y después del intercambio P2P para Ana y Carlos.

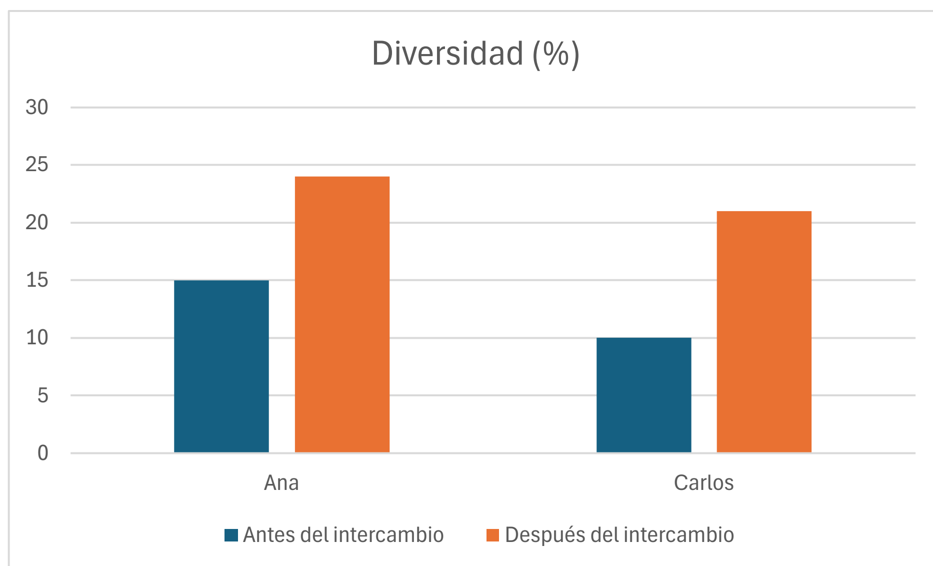


Figura 5.15: Comparación del índice de diversidad antes y después del intercambio P2P para Ana y Carlos.

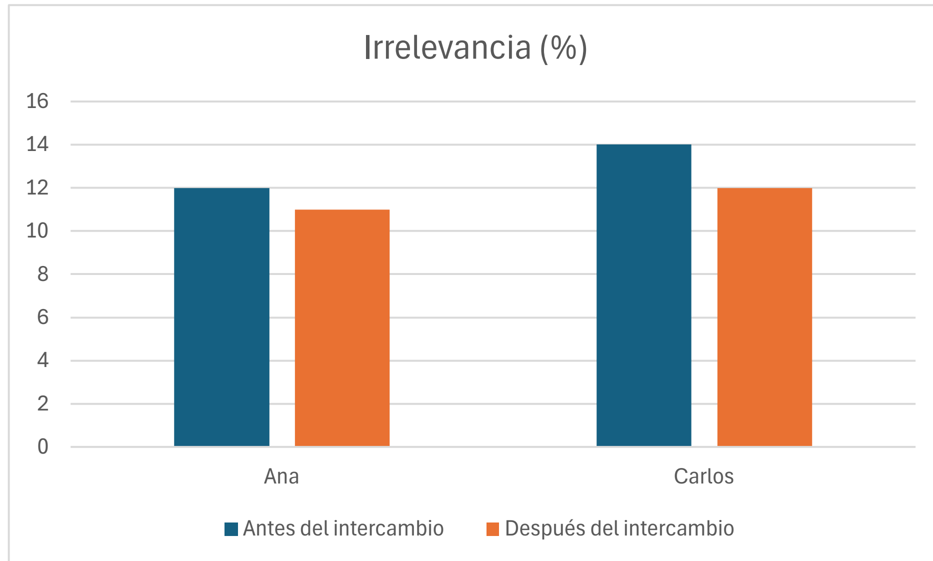


Figura 5.16: Comparación del índice de irrelevancia antes y después del intercambio P2P para Ana y Carlos.

5.4. Discusión general

Los resultados obtenidos a partir de las distintas pruebas realizadas permiten ofrecer una visión global del comportamiento y la eficiencia de los sistemas desarrollados. En conjunto, los experimentos confirman que las decisiones de diseño adoptadas en el proceso de modernización y descentralización han sido efectivas, garantizando tanto la estabilidad operativa como la viabilidad técnica de las soluciones propuestas.

En el caso de *R-Rules*, las pruebas de rendimiento del motor *Siddhi* demostraron que el sistema mantiene una latencia media estable incluso al aumentar el número de reglas activas, y que las variaciones en la latencia máxima son coherentes con la complejidad del conjunto de reglas evaluadas. Asimismo, la comparación entre la ejecución aislada del motor y la aplicación completa evidenció un incremento esperable en el consumo de CPU y memoria, sin comprometer la estabilidad general ni provocar comportamientos erráticos. Estas observaciones indican que la integración de *Siddhi* en la aplicación resulta eficiente y sostenible, a pesar de la sobrecarga inherente a un entorno más complejo.

En cuanto al módulo de comunicación *peer-to-peer* (P2P) incorporado en *R-Rules*, los resultados mostraron una latencia de transmisión moderada y una alta fiabilidad incluso ante reconexiones temporales, confirmando la robustez del mecanismo de intercambio directo. El sistema se comportó de forma estable bajo diferentes volúmenes de datos, manteniendo la integridad de la información transmitida y validando así la idoneidad del enfoque descentralizado para el intercambio de recomendaciones entre usuarios.

Por otro lado, la evaluación del sistema *PASEO* evidenció mejoras sustanciales derivadas del paso de una arquitectura cliente-servidor a una local distribuida. La reducción drástica de la latencia —de varios segundos en el modelo centralizado a decenas de milisegundos en la versión local— supone un avance significativo en términos de respuesta y experiencia de usuario. Aunque el procesamiento local incrementa el uso de CPU y RAM, estos valores se mantienen dentro de márgenes razonables y proporcionan un funcionamiento autónomo sin dependencia de la conectividad externa. Además, se observó que el rendimiento local varía en función de las características del hardware, reflejando el impacto del dispositivo en la eficiencia del sistema.

Finalmente, la evaluación del módulo *P2P* de *PASEO* demostró su capacidad para mejorar la calidad de las recomendaciones tras el intercambio entre nodos. Los incrementos registrados en los índices de precisión y diversidad, junto con la reducción del porcentaje de irrelevancia, confirman que la difusión de información entre pares enriquece la personalización sin introducir ruido significativo. Este comportamiento respalda la eficacia del enfoque colaborativo en la generación de recomendaciones contextualizadas.

En síntesis, los resultados de ambas evaluaciones ponen de manifiesto que la migración tecnológica y la adopción de una arquitectura local distribuida no solo mantienen la estabilidad y eficiencia del sistema original, sino que además amplían su rendimiento, autonomía y capacidad de adaptación. La combinación de procesamiento local y comunicación P2P se consolida así como una alternativa viable y ventajosa frente a las soluciones centralizadas tradicionales.

5.5. Conclusiones de la evaluación

La evaluación experimental realizada ha permitido validar de manera empírica los objetivos planteados en este trabajo. En primer lugar, se ha confirmado que la integración del motor de eventos *Siddhi* en *R-Rules* mantiene un rendimiento estable y predecible, garantizando la correcta ejecución de las reglas definidas y un consumo de recursos razonable, incluso bajo cargas elevadas. El módulo de comunicación *peer-to-peer* demostró ser eficiente y robusto, asegurando la transmisión fiable de información entre dispositivos y la continuidad de las operaciones ante reconexiones temporales.

Por otra parte, la comparación entre la arquitectura cliente-servidor original y la arquitectura local propuesta en *PASEO* evidenció claras ventajas de la descentralización. La versión local reduce significativamente la latencia, mejora la autonomía de los dispositivos y permite un procesamiento más equilibrado de las

tareas, manteniendo un consumo de recursos dentro de márgenes aceptables. Además, el módulo P2P incorporado en *PASEO* contribuye a la personalización y diversidad de las recomendaciones, sin introducir irrelevancia ni comprometer la calidad de los resultados.

La evaluación experimental no solo valida la viabilidad técnica de la migración y la arquitectura local distribuida, sino que también proporciona una base sólida para futuras optimizaciones y ampliaciones del sistema en entornos móviles colaborativos.

Capítulo 6

Conclusiones y líneas de trabajo futuro

6.1. Conclusiones personales

La realización de este Trabajo Fin de Grado ha supuesto para mí un reto exigente y, al mismo tiempo, una experiencia muy enriquecedora. Afrontar dos líneas de trabajo en paralelo —la migración tecnológica de **R-Rules** y el desarrollo de una implementación alternativa de **PASEO**— me permitió profundizar tanto en la modernización de software como en la adaptación de sistemas a entornos móviles.

En lo personal, considero que **R-Rules** me ayudó a comprender la importancia de mantener proyectos tecnológicos actualizados y sostenibles. Por otro lado, **PASEO** me permitió reflexionar sobre la eficiencia y la privacidad como factores clave en el diseño de soluciones reales para dispositivos móviles, además de introducirme en el concepto de comunicación *peer-to-peer* (P2P) como vía para fomentar la colaboración y la descentralización en sistemas de recomendación.

La incorporación de mecanismos P2P en ambos prototipos supuso un aprendizaje especialmente valioso, al demostrar cómo es posible compartir información útil entre dispositivos sin depender de servidores centrales, combinando autonomía, privacidad y cooperación entre usuarios.

Creo que este proyecto me ha permitido crecer no solo en lo técnico —mejorando mis competencias en Android, bases de datos, comunicación entre dispositivos y pruebas de rendimiento— sino también en lo personal, aprendiendo a organizar el tiempo, documentar decisiones y superar dificultades de manera autónoma.

La distribución aproximada del esfuerzo invertido en las distintas fases del TFG se muestra en la Tabla 6.1.

Tabla 6.1: Distribución estimada de horas de trabajo del TFG

Actividad	Horas estimadas	% sobre total
Investigación y revisión bibliográfica	45	9.9 %
Diseño de arquitectura R-Rules	35	7.7 %
Migración y modernización de R-Rules	80	17.5 %
Desarrollo de pruebas de latencia y rendimiento (R-Rules)	15	3.3 %
Integración y pruebas P2P en R-Rules	25	5.5 %
Análisis de resultados y gráficos (R-Rules)	10	2.2 %
Diseño de arquitectura PASEO	20	4.4 %
Implementación de PASEO local (Room, consultas)	60	13.1 %
Desarrollo de pruebas de latencia y rendimiento (PASEO)	40	8.8 %
Integración y pruebas P2P en PASEO	30	6.6 %
Análisis de resultados y gráficos (PASEO)	30	6.6 %
Redacción de capítulos de resultados y discusión	45	9.9 %
Redacción de introducción, conclusiones y anexos	25	5.5 %
Total	460	100 %

En resumen, pienso que este TFG ha sido una oportunidad para demostrar mi capacidad de afrontar proyectos complejos, consolidar conocimientos adquiridos a lo largo de la carrera y aportar una base sólida para futuros trabajos en el ámbito de la investigación y el desarrollo de software.

6.2. Conclusiones del proyecto

El presente Trabajo de Fin de Grado ha abordado dos ejes complementarios de evolución tecnológica: por un lado, la migración de **R-Rules**, un motor de reglas de recomendación originalmente implementado con tecnologías obsoletas; y, por otro, el desarrollo de **PASEO**, una implementación alternativa a los enfoques basados en SPARQL, concebida específicamente para funcionar en dispositivos móviles mediante el uso de **Room (SQLite)**.

En el caso de **R-Rules**, el proceso de migración permitió eliminar dependencias desfasadas y adaptar el sistema a un entorno moderno y sostenible, incrementando la mantenibilidad del código y facilitando su integración con arquitecturas actuales. Se trata de un resultado de gran relevancia dentro del proyecto, ya que asegura la continuidad de un motor de reglas con valor tanto práctico como académico, y sienta las bases para futuras ampliaciones y aplicaciones en contextos reales.

Por su parte, **PASEO** constituye una implementación alternativa al enfoque original basado en SPARQL. En lugar de mantener la dependencia de un motor

semántico externo, se optó por un diseño más ligero y adecuado a dispositivos móviles, fundamentado en **Room (SQLite)**. Una motivación esencial de este cambio fue reforzar la *privacidad*, evitando la necesidad de enviar información sensible de los usuarios a servicios externos. Al mismo tiempo, la solución permitió disponer de un sistema plenamente funcional sin recurrir a bibliotecas pesadas, reduciendo el tamaño de la aplicación y mejorando su rendimiento.

Las pruebas de latencia y uso de recursos (CPU y RAM) realizadas en distintos dispositivos y arquitecturas (local y cliente-servidor) confirmaron que la propuesta es viable, ofreciendo un equilibrio adecuado entre accesibilidad, rapidez y eficiencia en el manejo de recursos.

De forma complementaria, en ambos prototipos se incorporó un módulo de comunicación *peer-to-peer* (P2P), concebido para habilitar el intercambio directo de información entre dispositivos sin depender de un servidor central. En **R-Rules**, este componente permitió compartir actividades de forma distribuida, mientras que en **PASEO** posibilitó la propagación de valoraciones y elementos favoritos entre usuarios, introduciendo una forma ligera de colaboración local. Esta integración refuerza el carácter descentralizado de ambos sistemas y demuestra la viabilidad de combinar autonomía, privacidad y cooperación en entornos móviles.

En conjunto, el trabajo ha demostrado que es posible aplicar estrategias de migración tecnológica para extender la vida útil de sistemas existentes y adaptarlos a nuevos contextos de ejecución, sin necesidad de rediseños completos ni de sacrificar sus objetivos originales. Además, ha puesto de relieve la importancia de combinar decisiones técnicas con criterios de usabilidad y experiencia de usuario, especialmente en entornos móviles.

6.3. Limitaciones y trabajo futuro

El trabajo presenta, no obstante, algunas limitaciones. En el caso de **R-Rules**, la migración se centró en los aspectos básicos de modernización, sin explorar en detalle nuevas funcionalidades o integraciones con marcos de recomendación más avanzados. En **PASEO**, la eliminación del soporte SPARQL implica que ciertas consultas semánticas de mayor complejidad no puedan resolverse directamente, lo que restringe parcialmente la expresividad original del sistema.

Asimismo, las pruebas y desarrollos se han limitado al ecosistema **Android**, dejando fuera la evaluación de los mismos enfoques en **iOS**. Esta limitación es relevante, dado que el modelo de permisos, la gestión de recursos y las herramientas de desarrollo en iOS presentan diferencias significativas, lo que podría afectar tanto al rendimiento

como a la viabilidad de las soluciones implementadas.

De cara a trabajos futuros, la extensión de las aplicaciones *R-Rules* y *PASEO* hacia dispositivos iOS permitiría dotar al sistema de un carácter verdaderamente multiplataforma. Sin embargo, el análisis técnico de esta posibilidad revela desafíos significativos derivados tanto de las tecnologías empleadas como de los requisitos funcionales de ambas aplicaciones.

En primer lugar, *R-Rules* está desarrollada mayoritariamente en *React Native*, aunque incorpora un módulo nativo escrito en *Kotlin* para la gestión de funcionalidades específicas relacionadas con la comunicación entre dispositivos. Por su parte, *PASEO* se implementa íntegramente en Java, utilizando solo componentes propios del entorno Android. A pesar de esta diferencia tecnológica, ambas aplicaciones comparten un elemento crítico: la comunicación P2P basada en la API *Nearby Connections* de Google, la cual permite el descubrimiento cercano, el establecimiento de conexiones directas y el intercambio de datos entre dispositivos sin necesidad de infraestructura externa. Esta API es exclusiva de Android, por lo que no existe una portabilidad directa hacia iOS.

La ausencia de *Nearby Connections* en el entorno de Apple obliga a considerar alternativas como la API *Multipeer Connectivity*, que ofrece funcionalidades de comunicación local mediante Bluetooth y Wi-Fi, pero cuyo modelo operativo presenta diferencias significativas. Estas diferencias abarcan dimensiones como los protocolos de descubrimiento, la topología de red soportada, la estabilidad de la conexión y la gestión de sesiones simultáneas. En consecuencia, la migración multiplataforma no consiste en una simple adaptación del código existente, sino que requiere un rediseño completo del subsistema de comunicación distribuida.

Este rediseño implicaría, entre otros aspectos:

– La definición de una capa de abstracción que desacople la lógica de negocio del mecanismo concreto de transporte P2P.

– La implementación de módulos nativos diferenciados para Android y iOS, basados respectivamente en *Nearby Connections* y *Multipeer Connectivity*.

– La evaluación de la interoperabilidad entre dispositivos Android e iOS, ya que ninguna de estas APIs garantiza compatibilidad directa entre plataformas.

En el caso de *R-Rules*, el uso de *React Native* podría facilitar la reutilización de parte del código de la interfaz y de la lógica de alto nivel en una eventual versión para iOS. No obstante, el módulo nativo en *Kotlin* debería reescribirse en *Swift* u *Objective-C*, replicando su funcionalidad y ajustándolo al API de comunicación correspondiente. Para *PASEO*, cuyo código está completamente ligado al SDK de

Android y a bibliotecas puramente Java, el esfuerzo sería todavía mayor, ya que no existe una base multiplataforma que pueda reutilizarse directamente.

Por tanto, aunque frameworks como *React Native*, *Flutter* o *Ionic* pueden reducir el coste de portar componentes de la interfaz o parte de la lógica general, no eliminan la necesidad de reconstruir la capa de comunicación P2P, que constituye el núcleo técnico más complejo. Además, la validación final debería incluir pruebas exhaustivas en entornos reales, dado que las diferencias en la gestión de conexiones inalámbricas entre Android e iOS pueden afectar significativamente al rendimiento y la estabilidad del sistema.

En conclusión, la ampliación hacia iOS sería factible, pero implicaría un esfuerzo sustancial, especialmente en lo relativo al rediseño de la comunicación entre dispositivos y a la adaptación de los módulos nativos presentes en *R-Rules* y las dependencias específicas de Android empleadas en *PASEO*. Aun así, este trabajo permitiría dotar al sistema de una mayor escalabilidad y alcance, abriendo la puerta a un ecosistema verdaderamente multiplataforma.

En cuanto a líneas de trabajo futuras, se identifican varias direcciones prometedoras:

- Extender **R-Rules** con técnicas modernas de recomendación, como modelos basados en aprendizaje automático.

- Ampliar **PASEO** con funcionalidades avanzadas de análisis de contexto, integrando fuentes de datos externas o en tiempo real de manera eficiente.

- Optimizar aún más el rendimiento en dispositivos móviles, incorporando técnicas de caché o sincronización selectiva.

- Explorar la aplicabilidad de los enfoques empleados en otros dominios donde la migración tecnológica y la modernización de sistemas legados sean necesarias.

En conclusión, este trabajo no solo ha permitido actualizar y mejorar dos sistemas concretos, sino que ha mostrado un camino aplicable a otros proyectos de software que se enfrentan al reto de evolucionar en un ecosistema tecnológico en constante cambio.

Asimismo, la colaboración en este trabajo con investigadores del proyecto *NEAT-AMBIENCE* (PID2020-113037RB-I00, financiado por MICIU/AEI/10.13039/501100011033) ha resultado muy enriquecedora y ha permitido entrar en contacto con el ámbito de la investigación. El trabajo desarrollado en este TFG podría servir como contribución relevante en el contexto de dicho proyecto, ya que los prototipos y evaluaciones realizados ofrecen una base sólida para ser utilizados, ampliados o adaptados según se considere necesario.

De igual modo, existe potencial para la preparación futura de un artículo de investigación centrado en la comparación y evaluación de distintos despliegues y arquitecturas de sistemas de recomendación, siempre que los investigadores de *NEAT-AMBIENCE* o proyectos sucesivos lo consideren de interés y viabilidad.

Capítulo 7

Bibliografía

[1] Universidad de Zaragoza. Proyecto neat-ambience. https://webdiis.unizar.es/~silarri/NEAT-AMBIENCE/, 2025. 2021–2025, PID2020-113037RB-I00, financiado por MICIU/AEI/10.13039/501100011033. Último acceso: 8 de septiembre de 2025. Proyecto dirigido por Sergio Ilarri.

[2] Universidad de Zaragoza. R-rules: An approach for proactive mobile recommendations based on user-defined rules. https://webdiis.unizar.es/~silarri/prot/RRules/, 2025. Último acceso: 8 de septiembre de 2025. Proyecto NEAT-AMBIENCE.

[3] PASEO 1.0: Profile generation and content suggestion in the field of e-tourism. Proyecto de cooperación transfronteriza entre la Comunidad Autónoma de Aragón y la región francesa de Nueva Aquitania, financiado por el Gobierno de Aragón, 2018. Boletín Oficial de Aragón (BOA) núm. 140 del 20/07/2018. Duración: 20/07/2018–31/12/2018. Proyecto dirigido por Sergio Ilarri (Universidad de Zaragoza) y Philippe Roose (Universidad de Pau).

[4] PASEO 2.0: Profile generation and content suggestion in the field of e-tourism. Proyecto de cooperación transfronteriza entre la Comunidad Autónoma de Aragón y la región francesa de Nueva Aquitania, financiado por el Gobierno de Aragón, 2021. Boletín Oficial de Aragón (BOA) núm. 118 del 02/06/2021. Duración: 02/06/2021–15/11/2021. Proyecto dirigido por Sergio Ilarri (Universidad de Zaragoza) y Philippe Roose (Universidad de Pau).

[5] Gediminas Adomavicius, Bamshad Mobasher, Francesco Ricci, and Alex Tuzhilin. Context-aware recommender systems. *AI Magazine*, 32(3):67–80, 2011.

[6] Sergio Ilarri and Raquel Trillo-Lado. An approach for proactive mobile recommendations based on user-defined rules. *Expert Systems with Applications*, 242:122714:1–122714:22, May 2024.

[7] Bo Zhang and S. Shyam Sundar. Proactive vs. reactive personalization: Can customization of privacy enhance user experience? *International Journal of Human-Computer Studies*, 128:86–99, 2019.

[8] Waqar Ali, Rajesh Kumar, Zhiyi Deng, Yansong Wang, and Jie Shao. A federated learning approach for privacy protection in context-aware recommender systems. *The Computer Journal*, 64(7):1016–1027, 04 2021.

[9] Christos Chronis, Iraklis Varlamis, Yassine Himeur, Aya N. Sayed, Tamim M. AL-Hasan, Armstrong Nhlabatsi, Faycal Bensaali, and George Dimitrakopoulos. A survey on the use of federated learning in privacy-preserving recommender systems. *IEEE Open Journal of the Computer Society*, 5:227–247, 2024.

[10] María del Carmen Rodríguez-Hernández, Sergio Ilarri, Raquel Trillo, and Ramón Hermoso. Context-aware recommendations using mobile p2p. In *Proceedings of the 15th International Conference on Advances in Mobile Computing & Multimedia*, MoMM2017, page 82–91, New York, NY, USA, 2017. Association for Computing Machinery.

[11] Ramón Hermoso, Sergio Ilarri, Raquel Trillo, and María del Carmen Rodríguez-Hernández. Push-based recommendations in mobile computing using a multi-layer contextual approach. In *Proceedings of the 13th International Conference on Advances in Mobile Computing and Multimedia*, MoMM 2015, page 149–158, New York, NY, USA, 2015. Association for Computing Machinery.

[12] Ivens Portugal, Paulo Alencar, and Donald Cowan. The use of machine learning algorithms in recommender systems: A systematic review. *Expert Systems with Applications*, 97:205–227, 2018.

[13] María del Carmen Rodríguez-Hernández, Sergio Ilarri, Ramón Hermoso, and Raque Trillo-Lado. Towards trajectory-based recommendations in museums: Evaluation of strategies using mixed synthetic and real data. *Procedia Computer Science*, 113:234–239, 2017. The 8th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2017) / The 7th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2017) / Affiliated Workshops.

[14] Panagiotis Prattis. Recommendation system for android java app that finds an ideal destination with the knn algorithm. `https://github.com/pprattis/Recommendation-System-for-Android-Java-App-that-finds-an-ideal-des`

tination-with-the-kNN-Algorithm. GitHub repository. Último acceso: 14 septiembre 2025.

[15] Lakshya Karwa. Movie recommender app. https://github.com/LakshyaKarwa/Movie-Recommender-App. GitHub repository. Último acceso: 14 septiembre 2025.

[16] TensorFlow. Tensorflow lite guide. https://www.tensorflow.org/lite/guide?hl=es-419. Último acceso: 14 septiembre 2025.

[17] Google Developers. Room persistence library. https://developer.android.com/training/data-storage/room?hl=es-419. Último acceso: 14 septiembre 2025.

[18] MongoDB Inc. Realm database documentation. https://www.mongodb.com/docs/realm/. Último acceso: 14 septiembre 2025.

[19] React Native. Set up your environment. https://reactnative.dev/docs/set-up-your-environment. Documentación oficial. Último acceso: 14 septiembre 2025.

[20] David Hedenus. Mobile rdf. https://www.hedenus.de/rdf/. Último acceso: 14 septiembre 2025.

[21] Sergio Ilarri, Pedro Roig, and Raquel Trillo. Geosprings: Towards a location-aware mobile agent platform. In Miguel R. Luaces and Farid Karimipour, editors, *Web and Wireless Geographical Information Systems*, pages 51–60, Cham, 2018. Springer International Publishing.

[22] Bluetooth SIG. Bluetooth core specification v5.4. https://www.bluetooth.com/specifications/bluetooth-core-specification/, 2023. Último acceso: 31 de octubre de 2025.

[23] Google. Webrtc overview. https://webrtc.org/, 2024. Último acceso: 31 de octubre de 2025.

[24] Google Developers. Nearby connections api overview. https://developers.google.com/nearby/connections/overview?hl=es-419, 2025. Último acceso: 16 de octubre de 2025. Documentación oficial de Google Developers.

[25] Carlos Bobed, Roberto Yus, Fernando Bobillo, and Eduardo Mena. Semantic reasoning on mobile devices: Do androids dream of efficient reasoners? *Journal of Web Semantics*, 35:167–183, 2015.

[26] Amit Livne, Eliad Shem Tov, Adir Solomon, Achiya Elyasaf, Bracha Shapira, and Lior Rokach. Evolving context-aware recommender systems with users in mind. *Expert Systems with Applications*, 189:116042, 2022.

[27] Sergio Ilarri, Irene Fumanal, and Raquel Trillo-Lado. An experience with the implementation of a rule-based triggering recommendation approach for mobile devices. In *The 23rd International Conference on Information Integration and Web Intelligence*, iiWAS2021, page 562–570, New York, NY, USA, 2022. Association for Computing Machinery.

Lista de Figuras

3.1. Diagrama de secuencia del flujo de información P2P entre dispositivos en *R-Rules*. 26

4.1. Flujo de acceso a la API SPARQL con integración local 42
4.2. Diagrama de secuencia del flujo de información P2P entre dispositivos en *Paseo*. 46

5.1. Resultados de las pruebas de rendimiento en dispositivo Android con Siddhi. 57
5.2. Comparativa de latencias entre Siddhi aislado y la aplicación completa. 59
5.3. Consumo de CPU en Siddhi aislado y en la aplicación completa. 59
5.4. Consumo de memoria RSS en Siddhi aislado y en la aplicación completa. 60
5.5. Consumo de memoria PSS en Siddhi aislado y en la aplicación completa. 61
5.6. Latencia total de transmisión en función del número de actividades enviadas en bloque JSON. 64
5.7. Evolución de la velocidad efectiva en función del tamaño del número de actividades enviadas en bloque JSON. 64
5.8. Comparativa de la latencia original y tras reconexión para diferentes tamaños de bloque de actividades. 65
5.9. Resultados de latencia en el Xiaomi Redmi Note 10S para la arquitectura local. 69
5.10. Resultados de latencia en el Xiaomi Redmi Note 10S para la arquitectura cliente-servidor. 70
5.11. Resultados de latencia en arquitectura local para el dispositivo Vivo Y36. 71
5.12. Resultados de latencia en arquitectura cliente-servidor para el dispositivo Vivo Y36. 71
5.13. Comparación de las medias de CPU, PSS y RSS entre dispositivos y arquitecturas. 74
5.14. Comparación del índice de precisión antes y después del intercambio P2P para Ana y Carlos. 82

5.15. Comparación del índice de diversidad antes y después del intercambio P2P para Ana y Carlos. 82
5.16. Comparación del índice de irrelevancia antes y después del intercambio P2P para Ana y Carlos. 83

A.1. Esquema relacional de la base de datos de PASEO 104

F.1. Interacción básica con las recomendaciones en la pantalla principal. . . 134
F.2. Gestión de recomendaciones marcadas como favoritas. 135
F.3. Pantalla Historic. 136
F.4. Menú lateral de navegación. 137
F.5. Pantalla Profile. 138
F.6. Pantalla Settings de la aplicación. 139
F.7. Gestión de Triggering Rules. 141
F.8. Confirmación de eliminación de la Triggering Rule 142
F.9. Creación de Triggering Rule (parte 1). 143
F.10. Confirmación de creación exitosa de la Triggering Rule 144
F.11. Pantalla principal de Exclusion Sets. 145
F.12. Creación de un nuevo Exclusion Set. 146
F.13. Confirmación de creación de Exclusion Set. 146
F.14. Detalle de un Exclusion Set. 147
F.15. Creación de regla contextual. 148
F.16. Visualización de reglas de contexto. 149
F.17. Pantalla inicial de registro de usuario en PASEO 152
F.18. Pantalla inicial con recomendaciones de puntos de interés cercanos . . . 153
F.19. Pantallas de detalle de Puntos de Interés (PIs) en PASEO 154
F.20. Menú principal de navegación lateral 155
F.21. Pantalla de ajustes de la aplicación 156
F.22. Pantalla de Puntos de Interés valorados por el usuario 157
F.23. Pantalla informativa: Acerca de PASEO 158

Lista de Tablas

2.1. Comparativa de tecnologías P2P analizadas 10

3.1. Comparativa entre el entorno de desarrollo anterior y el actualizado tras la migración 15

3.2. Resumen de tecnologías utilizadas en R-Rules 28

4.1. Comparativa de alternativas exploradas para sustituir el motor Virtuoso en la arquitectura local de PASEO 37

4.2. Consulta SPARQL para obtener el número total de PIs en el endpoint del Ayuntamiento de Zaragoza 39

4.3. Comparación entre la arquitectura cliente-servidor original y la nueva arquitectura local en dispositivo 48

4.4. Resumen de tecnologías utilizadas en PASEO 52

5.1. Resultados de latencia en el Xiaomi Redmi Note 10S (ejecución en local). 69

5.2. Resultados de latencia en el Xiaomi Redmi Note 10S (arquitectura cliente-servidor). 69

5.3. Resultados de latencia en el Vivo Y36 (ejecución en local). 70

5.4. Resultados de latencia en el Vivo Y36 (arquitectura cliente-servidor). . 71

5.5. Uso de CPU y memoria en el Xiaomi Redmi Note 10S. 73

5.6. Uso de CPU y memoria en el Vivo Y36. 74

5.7. Resultados experimentales del intercambio P2P. 80

6.1. Distribución estimada de horas de trabajo del TFG 87

Anexos

Anexos A

PASEO: esquema relacional de la base de datos

El sistema PASEO se apoya en una base de datos relacional en SQLite, diseñada para almacenar la información de los lugares, usuarios y las interacciones entre ambos (valoraciones, favoritos y recomendaciones). Además, incluye tablas auxiliares para manejar información en formato JSON y métricas de similitud entre lugares, utilizadas en los algoritmos de recomendación.

En la Figura A.1 se presenta el esquema general de la base de datos, donde se aprecia la tabla central `PLACE`, sobre la que se relacionan el resto de entidades.

El diseño de la base de datos sigue una estructura sencilla y normalizada, en la que:

- `USER` gestiona los datos de los usuarios de la aplicación.
- `PLACE` almacena los lugares disponibles, con sus atributos básicos (nombre, coordenadas, tipo).
- Tablas de relación como `VALORATION` y `FAVOURITE` vinculan usuarios con lugares, registrando valoraciones y marcadores de favoritos.
- `COSINE` y `RECOMMENDATION` recogen la lógica de recomendación basada en similitud entre lugares y predicciones para usuarios.
- `JSON` sirve para almacenar información adicional en formato semiestructurado.

Este esquema proporciona una base de datos ligera pero suficiente para gestionar la información necesaria en PASEO y dar soporte a las funcionalidades de recomendación y personalización del sistema.

Table: COSINE			
#	**idPlace1** INTEGER	**idPlace2** INTEGER	**cosine** DOUBLE

Table: FAVOURITE			
#	**idUser** INTEGER	**idPlace** INTEGER	**favourite** BOOLEAN

Table: JSON		
#	**idPlace** INTEGER	**json** TEXT

Table: PLACE						
#	**idPlace** INTEGER	**name** TEXT	**comment** TEXT	**longitud** TEXT	**latitud** TEXT	**tipo** TEXT

Table: RECOMMENDATION						
#	**idUser** INTEGER	**idPlace** INTEGER	**rating** DOUBLE	**simCosine** DOUBLE	**producto** DOUBLE	**motivo** STRING

Table: USER				
#	**id** INTEGER	**name** STRING	**password** TEXT	**google** INTEGER

Table: VALORATION				
#	**idUser** INTEGER	**idPlace** INTEGER	**numValoration** DOUBLE	**timestamp** TEXT

Document generated by SQLiteStudio v3.4.17 on Fri Jun 27 07:44:50 2025

Figura A.1: Esquema relacional de la base de datos de PASEO

Anexos B

Implementación del worker de actualización de la base de datos

El siguiente fragmento de código muestra la configuración del `WorkManager` para ejecutar la actualización periódica de la base de datos local de Zaragoza cada semana:

```
WorkManager workManager = WorkManager.getInstance(context);
PeriodicWorkRequest updateRequest =
        new PeriodicWorkRequest.Builder(
                UpdateZaragozaPlacesWorker.class,
                7, TimeUnit.DAYS // cada semana
        ).build();

workManager.enqueueUniquePeriodicWork(
        "updateZaragozaPlaces",
        ExistingPeriodicWorkPolicy.KEEP,
        updateRequest
);
```

Para la implementación completa se utiliza la biblioteca **Android WorkManager**, que permite ejecutar tareas en segundo plano de manera confiable incluso si la aplicación no está activa.

El worker `UpdateZaragozaPlacesWorker` realiza las siguientes funcionalidades principales:

- Consulta el endpoint SPARQL del Ayuntamiento de Zaragoza para obtener los Puntos de Interés más recientes.
- Inserta los registros nuevos en la base de datos local `Room`, y elimina aquellos eventos o registros que ya no son vigentes, garantizando consistencia de los datos.
- Se ejecuta automáticamente cada semana, evitando duplicaciones mediante `enqueueUniquePeriodicWork`.

El diseño de la base de datos y la lógica completa de inicialización se encuentra en la clase `DatabaseClient`. Esta clase se encarga de crear y configurar la base de datos local mediante Room, cargar los datos iniciales en caso de que las tablas estén vacías, y establecer la ejecución del worker periódico de actualización. A continuación se incluye el código relevante del callback encargado de esta inicialización:

```java
.addCallback(new RoomDatabase.Callback() {
    @Override
    public void onOpen(@NonNull SupportSQLiteDatabase db) {
        super.onOpen(db);

        Executors.newSingleThreadExecutor().execute(() -> {
            ZaragozaPlaceDao zaragozaPlaceDao = database.zaragozaPlaceDao();

            if (zaragozaPlaceDao.getAll().isEmpty()) {
                Log.i("DatabaseClient", "Tabla vacía, cargando lugares desde
                 → SPARQL...");
                InsertDatabase.insertInitialZaragozaPlaces(zaragozaPlaceDao);
            } else {
                Log.i("DatabaseClient", "Tabla ZaragozaPlace ya tiene datos, no se
                 → insertan iniciales.");
            }

            // Lanzar el worker periódico
            WorkManager workManager = WorkManager.getInstance(context);
            PeriodicWorkRequest updateRequest =
                    new PeriodicWorkRequest.Builder(
                            UpdateZaragozaPlacesWorker.class,
                            7, TimeUnit.DAYS // cada semana
                    ).build();

            workManager.enqueueUniquePeriodicWork(
                    "updateZaragozaPlaces",
                    ExistingPeriodicWorkPolicy.KEEP, // evita duplicaciones
                    updateRequest
            );
        });
    }
})
```

Anexos C

Tests de latencia implementados en PASEO

En este anexo se incluyen los tests utilizados para la medición de latencia en las dos arquitecturas evaluadas de PASEO:

– **RecommendationLatencyTest**: mide el tiempo de respuesta en la arquitectura local, ejecutando las recomendaciones directamente sobre la base de datos en el dispositivo.

– **RecommendationLatencyServerTest**: mide el tiempo de respuesta en la arquitectura cliente-servidor, enviando las peticiones al backend mediante llamadas HTTP.

Prueba de latencia en local

```
1  @RunWith(AndroidJUnit4.class)
2  public class RecommendationLatencyTest {
3
4      @Test
5      public void testRecommendationLatencyGroups() {
6          Context context =
            ↪ InstrumentationRegistry.getInstrumentation().getTargetContext();
7          RecommendCloseness recommender = new RecommendCloseness(context);
8
9          // Parámetros por grupo
10         double[] longitudes = {-0.8790, -0.8775, -0.8750, -0.8805, -0.8820,
            ↪ -0.8780, -0.8765, -0.8810, -0.8830, -0.8795};
11         double[] latitudes = {41.6420, 41.6450, 41.6480, 41.6500, 41.6550,
            ↪ 41.6580, 41.6600, 41.6630, 41.6660, 41.6690};
12         double[] maxDistances = {500, 800, 1000, 1200, 1500, 1800, 2000, 2500,
            ↪ 2800, 3000};
13
14
15         int nGrupos = longitudes.length; // 10 grupos
16         int repeticionesPorGrupo = 10;
17
18         for (int g = 0; g < nGrupos; g++) {
19             double longitude = longitudes[g];
```

```
20              double latitude = latitudes[g];
21              double maxDistance = maxDistances[g];
22
23              List<Long> latencies = new ArrayList<>();
24
25              for (int i = 0; i < repeticionesPorGrupo; i++) {
26                  long start = System.nanoTime();
27                  recommender.getRecommendations(longitude, latitude, maxDistance,
                    ↪  10);
28                  long end = System.nanoTime();
29
30                  long latencyMs = (end - start) / 1_000_000;
31                  latencies.add(latencyMs);
32
33                  Log.d("LatencyTest", "Grupo " + (g+1) + " - Ejecución " + (i+1) +
                    ↪  ": " + latencyMs + " ms");
34              }
35
36              long min = latencies.stream().min(Long::compare).orElse(-1L);
37              long max = latencies.stream().max(Long::compare).orElse(-1L);
38              double avg =
                ↪  latencies.stream().mapToLong(Long::longValue).average().orElse(-1);
39
40              Log.d("LatencyTest", "Grupo " + (g+1) + " -> Min: " + min + " ms, Max:
                ↪  " + max + " ms, Media: " + avg + " ms");
41          }
42      }
43  }
```

Prueba de latencia contra el servidor

```
1   @RunWith(AndroidJUnit4.class)
2   public class RecommendationLatencyServerTest {
3
4       @Test
5       public void testRecommendationLatencyGroups_Server() throws Exception {
6           int nGrupos = 10;
7           int repeticionesPorGrupo = 10;
8
9           String ip = "127.0.0.1";
10          int port = 8080;
11          String url = "http://" + ip + ":" + port +
            ↪  "/PaseoServlets/servlet/RecommendCloseness";
12          String maxItems = "10";
13
14          // Parámetros por grupo dentro de Zaragoza
15          String[] longitudes = {"-0.8790", "-0.8775", "-0.8750", "-0.8805",
            ↪  "-0.8820", "-0.8780", "-0.8765", "-0.8810", "-0.8830", "-0.8795"};
16          String[] latitudes  = {"41.6420", "41.6450", "41.6480", "41.6500",
            ↪  "41.6550", "41.6580", "41.6600", "41.6630", "41.6660", "41.6690"};
17          String[] maxDistances = {"500", "800", "1000", "1200", "1500", "1800",
            ↪  "2000", "2500", "2800", "3000"};
18
19          for (int g = 0; g < nGrupos; g++) {
20              String longitude = longitudes[g];
21              String latitude = latitudes[g];
22              String maxDistance = maxDistances[g];
```

```
23
24              List<Long> latencies = new ArrayList<>();
25
26              for (int i = 0; i < repeticionesPorGrupo; i++) {
27                  long start = System.nanoTime();
28
29                  HttpClient httpClient = new DefaultHttpClient();
30                  HttpPost httpPost = new HttpPost(url);
31
32                  List<BasicNameValuePair> list = new ArrayList<>();
33                  list.add(new BasicNameValuePair("latitud", latitude));
34                  list.add(new BasicNameValuePair("longitud", longitude));
35                  list.add(new BasicNameValuePair("maxDistance", maxDistance));
36                  list.add(new BasicNameValuePair("maxItems", maxItems));
37                  httpPost.setEntity(new UrlEncodedFormEntity(list));
38
39                  HttpResponse httpResponse = httpClient.execute(httpPost);
40                  HttpEntity httpEntity = httpResponse.getEntity();
41                  String s = EntityUtils.toString(httpEntity, HTTP.UTF_8);
42
43                  long end = System.nanoTime();
44                  long latencyMs = (end - start) / 1_000_000;
45                  latencies.add(latencyMs);
46
47                  Log.d("LatencyTestServer", "Grupo " + (g+1) + " - Ejecución " +
                     ↪ (i+1) + ": " + latencyMs + " ms, resp=" + s);
48              }
49
50              long min = latencies.stream().min(Long::compare).orElse(-1L);
51              long max = latencies.stream().max(Long::compare).orElse(-1L);
52              double avg =
                 ↪ latencies.stream().mapToLong(Long::longValue).average().orElse(-1);
53
54              Log.d("LatencyTestServer", "Grupo " + (g+1) + " -> Min: " + min + "
                 ↪ ms, Max: " + max + " ms, Media: " + avg + " ms");
55          }
56      }
57  }
```

Anexos D

Medición de métricas de recursos

Script en Bash para captura de métricas

El siguiente script obtiene métricas de uso de CPU y memoria de la aplicación Android y las guarda en un archivo CSV:

```
#!/bin/bash

# Nombre del paquete
PACKAGE_NAME="es.unizar.eina.paseo"

# Número de muestras y frecuencia (segundos entre mediciones)
SAMPLES=30
INTERVAL=2

# Archivo de salida
OUTPUT_FILE="app_metrics.csv"

echo "timestamp,cpu%,rss_kb,pss_kb" > $OUTPUT_FILE

for i in $(seq 1 $SAMPLES)
do
    TIMESTAMP=$(date +"%Y-%m-%d %H:%M:%S")

    # Obtener PID de la app
    PID=$(adb shell pidof $PACKAGE_NAME)

    if [ -z "$PID" ]; then
        echo "$TIMESTAMP,App no en ejecución" >> $OUTPUT_FILE
    else
        # CPU (%)
        CPU=$(adb shell top -b -n 1 -p $PID | grep $PID | awk '{print $9}' | tr -d
         ↪ '%')

        # RSS en KB (memoria total residente)
        RSS=$(adb shell dumpsys meminfo $PACKAGE_NAME | grep "TOTAL " | awk
         ↪ '{print $2}')

        # PSS total en KB
        PSS=$(adb shell dumpsys meminfo $PACKAGE_NAME | grep "TOTAL PSS" | awk
         ↪ '{print $3}')

        echo "$TIMESTAMP,$CPU,$RSS,$PSS" >> $OUTPUT_FILE
```

```
    fi

    sleep $INTERVAL
done

echo "Datos guardados en $OUTPUT_FILE"
```

Script en Python para procesar métricas

Este script lee el archivo CSV generado por el anterior, calcula estadísticas (media, mínimo y máximo) y muestra los resultados:

```python
import csv

# Nombre del archivo CSV
csv_file = "app_metrics.csv"

# Listas para almacenar valores
cpu_list = []
rss_list = []
pss_list = []

# Variable para "guardar" PSS cuando aparece en línea separada
pending_pss = None

with open(csv_file, newline='') as f:
    reader = csv.reader(f)
    next(reader)  # Saltar cabecera

    for row in reader:
        if len(row) >= 3 and row[0] != "PSS:":
            timestamp, cpu, rss = row[0], row[1], row[2]
            cpu_val = float(cpu)
            rss_val = int(rss)

            if pending_pss is not None:
                pss_val = int(pending_pss)
                pending_pss = None
            else:
                pss_val = None

            cpu_list.append(cpu_val)
            rss_list.append(rss_val)
            if pss_val is not None:
                pss_list.append(pss_val)

        elif row[0] == "PSS:":
            pending_pss = row[1]

def stats(values):
    return {
        "media": sum(values)/len(values),
        "min": min(values),
        "max": max(values)
    }

```

```
45  cpu_stats = stats(cpu_list)
46  rss_stats = stats(rss_list)
47  pss_stats = stats(pss_list) if pss_list else None
48
49  print("CPU%      → Media: {:.2f}%, Mín: {:.2f}%, Máx:
     ↪ {:.2f}%".format(cpu_stats["media"], cpu_stats["min"], cpu_stats["max"]))
50  print("RSS KB    → Media: {:.0f} KB, Mín: {} KB, Máx: {}
     ↪ KB".format(rss_stats["media"], rss_stats["min"], rss_stats["max"]))
51  if pss_stats:
52      print("PSS KB    → Media: {:.0f} KB, Mín: {} KB, Máx: {}
         ↪ KB".format(pss_stats["media"], pss_stats["min"], pss_stats["max"])_
```

Anexos E

Implementaciones del módulo P2P en R-Rules y PASEO

En este anexo se describe la implementación del servicio responsable de gestionar la comunicación *peer-to-peer* en las aplicaciones *R-Rules* y *Paseo*. Ambas versiones comparten la misma estructura funcional y utilizan la API *Nearby Connections* de Android para el descubrimiento, conexión y transmisión de datos entre dispositivos cercanos.

La principal diferencia entre ambas reside en el lenguaje de programación empleado: la implementación original de *R-Rules* está desarrollada en `Kotlin`, mientras que la versión integrada en *Paseo* se ha adaptado a `Java` para ajustarse a la arquitectura y dependencias específicas de la aplicación.

A continuación se resumen los componentes fundamentales del servicio:

- **Inicialización y ciclo de vida:** el servicio puede iniciarse mediante el método `start()`, que lanza en paralelo los procesos de *advertising* y *discovery*. Para evitar estados inactivos prolongados, se incluye un mecanismo de *watchdog* que relanza periódicamente el descubrimiento aplicando una política de *backoff exponencial*.

- **Gestión de conexiones:** el `ConnectionLifecycleCallback` se encarga de aceptar automáticamente las conexiones entrantes, notificar los eventos de conexión y reconexión, y reiniciar el descubrimiento en caso de desconexión. Los identificadores de los dispositivos conectados se mantienen en una lista interna para garantizar la coherencia del estado de red.

- **Descubrimiento de dispositivos:** el `EndpointDiscoveryCallback` permite detectar nodos cercanos y establecer la conexión de manera automática. Si un dispositivo deja de estar disponible, el servicio reinicia el proceso de descubrimiento para mantener la resiliencia de la red.

– **Transmisión de datos:** la comunicación se realiza mediante objetos `Payload`. El método `sendData()` envía mensajes en formato de texto a todos los nodos conectados, mientras que el `PayloadCallback` gestiona los datos recibidos e invoca las funciones de `callback` registradas por la aplicación.

– **Robustez:** el servicio implementa un cierre controlado de los procesos de *advertising*, *discovery* y de las conexiones activas a través del método `stop()`, previniendo fugas de recursos y asegurando un comportamiento estable y predecible.

En ambas implementaciones, la estructura lógica del servicio es equivalente. La versión en *Paseo*, escrita en `Java`, mantiene el mismo flujo de trabajo descrito anteriormente, adaptando únicamente la sintaxis y ciertos detalles de gestión de hilos y *callbacks* para ajustarse a las convenciones del lenguaje y a la arquitectura interna de la aplicación.

E.1. Implementación en R-Rules (Kotlin)

En la Figura 3.1 del cuerpo principal se muestra un diagrama de secuencia que ilustra este flujo, mientras que el código recoge la implementación íntegra del servicio. El código del servicio se puede consultar a continuación

```
1  package com.carsproject.p2p
2
3  import android.Manifest
4  import android.content.pm.PackageManager
5  import android.os.Build
6  import android.util.Log
7  import androidx.core.content.ContextCompat
8  import com.google.android.gms.nearby.Nearby
9  import com.google.android.gms.nearby.connection.*
10 import android.os.Handler
11 import android.os.Looper
12
13 class P2PService(private val context: android.content.Context) {
14
15     private val handler = Handler(Looper.getMainLooper())
16     private val connectionsClient = Nearby.getConnectionsClient(context)
17     private val strategy = Strategy.P2P_CLUSTER
18     private var connectedEndpoints: MutableSet<String> = mutableSetOf()
19     // Estado interno
20     private var running = false
21     //Callback cuando llegan datos
22     var onDataReceived: ((String) -> Unit)? = null
23     var onConnected: ((String) -> Unit)? = null
24
25     // Valores configurables
26     private val rediscoveryIntervalInitial = 30_000L   // 30s iniciales
```

```
27      private val rediscoveryMaxDelay = 5 * 60_000L       // 5 minutos máximo
28      private var currentRediscoveryDelay = rediscoveryIntervalInitial
29
30      // Guardar los valores con los que arrancamos para relanzarlos
31      private var storedServiceName: String? = null
32      private var storedUserId: String? = null
33
34      // Runnable que relanza discovery periódicamente (watchdog)
35      private val rediscoveryRunnable = object : Runnable {
36          override fun run() {
37              if (running) {
38                  Log.d("P2PService", "Watchdog: forzando discovery
                    ↪  (delay=${currentRediscoveryDelay})")
39                  // Llamamos a restartDiscovery para gestionar backoff y restart
                    ↪  seguro
40                  restartDiscovery()
41                  // reprogramar próximo chequeo según backoff actual
42                  handler.postDelayed(this, currentRediscoveryDelay)
43              }
44          }
45      }
46
47      //Iniciar Advertising + Discovery
48      fun start(serviceName: String, userId: String) {
49          // Guardamos para relanzamientos posteriores
50          storedServiceName = serviceName
51          storedUserId = userId
52
53          // Iniciar Advertising + Discovery inicial
54          startAdvertising(serviceName, userId)
55          startDiscovery(serviceName)
56          running = true
57
58          // Reset del backoff al arrancar
59          currentRediscoveryDelay = rediscoveryIntervalInitial
60
61          // Iniciar watchdog que reintentará discovery periódicamente
62          handler.postDelayed(rediscoveryRunnable, currentRediscoveryDelay)
63      }
64
65      fun stop() {
66          try {
67              connectionsClient.stopAdvertising()
68              connectionsClient.stopDiscovery()
69              connectionsClient.stopAllEndpoints()
70          } catch (e: Exception) {
71              Log.w("P2PService", "Error al parar Nearby: ${e.message}")
72          }
73
74          connectedEndpoints.clear()
75          Log.d("P2PService", "Servicio parado")
76          running = false
77
78          // limpiar watchdog
79          handler.removeCallbacks(rediscoveryRunnable)
80          // reset backoff
81          currentRediscoveryDelay = rediscoveryIntervalInitial
82      }
```

```
    fun isStopped(): Boolean {
        return !running
    }

    //Enviar datos a todos los conectados
    fun sendData(data: String) {
        val payload = Payload.fromBytes(data.toByteArray())
        connectedEndpoints.forEach { endpointId ->
            connectionsClient.sendPayload(endpointId, payload)
        }
    }

    // ====================
    // Nearby API Callbacks
    // ====================

    private fun startAdvertising(serviceName: String, userId: String) {
        val advertisingOptions =
         → AdvertisingOptions.Builder().setStrategy(strategy).build()
        connectionsClient.startAdvertising(
            userId,  // nombre del endpoint
            serviceName,
            connectionLifecycleCallback,
            advertisingOptions
        ).addOnSuccessListener {
            Log.d("P2PService", "Advertising iniciado")
        }.addOnFailureListener {
            Log.e("P2PService", "Error en advertising: £{it.message}")
        }
    }

    private fun startDiscovery(serviceName: String) {
        val discoveryOptions =
         → DiscoveryOptions.Builder().setStrategy(strategy).build()
        connectionsClient.startDiscovery(
            serviceName,
            endpointDiscoveryCallback,
            discoveryOptions
        ).addOnSuccessListener {
            Log.d("P2PService", "Discovery iniciado")
            // éxito -> reset del backoff
            currentRediscoveryDelay = rediscoveryIntervalInitial
        }.addOnFailureListener {
            Log.e("P2PService", "Error en discovery: £{it.message}")
        }
    }

    //Descubrimiento de dispositivos
    private val endpointDiscoveryCallback = object : EndpointDiscoveryCallback() {
        override fun onEndpointFound(endpointId: String, info:
         → DiscoveredEndpointInfo) {
            Log.d("P2PService", "Dispositivo encontrado: £endpointId
             → (£{info.endpointName})")
            // Conectar automáticamente
```

```
137             connectionsClient.requestConnection("P2PDevice", endpointId,
                 ↪  connectionLifecycleCallback)
138         }
139
140         override fun onEndpointLost(endpointId: String) {
141             Log.d("P2PService", "Dispositivo perdido: $endpointId")
142             connectedEndpoints.remove(endpointId)
143             // relanzar discovery para volver a encontrar peers
144             restartDiscovery()
145         }
146     }
147
148     // Reinicio del descubrimiento
149     private fun restartDiscovery(immediate: Boolean = false) {
150         // Evitar llamadas concurrentes
151         try {
152             connectionsClient.stopDiscovery()
153         } catch (e: Exception) {
154             Log.w("P2PService", "stopDiscovery fallo: ${e.message}")
155         }
156
157         val delay = if (immediate) 0L else currentRediscoveryDelay
158         handler.postDelayed({
159             storedServiceName?.let { sName ->
160                 startDiscovery(sName)
161             } ?: Log.w("P2PService", "No storedServiceName; no se puede relanzar
                 ↪  discovery")
162         }, delay)
163
164         // aumentar backoff para la próxima vez (doble hasta el máximo)
165         currentRediscoveryDelay = (currentRediscoveryDelay *
             ↪  2).coerceAtMost(rediscoveryMaxDelay)
166     }
167
168     // Ciclo de vida de la conexión
169     private val connectionLifecycleCallback = object :
         ↪  ConnectionLifecycleCallback() {
170         override fun onConnectionInitiated(endpointId: String, connectionInfo:
             ↪  ConnectionInfo) {
171             Log.d("P2PService", "Conexión iniciada con $endpointId")
172             // Aceptar automáticamente
173             connectionsClient.acceptConnection(endpointId, payloadCallback)
174         }
175
176         override fun onConnectionResult(endpointId: String, result:
             ↪  ConnectionResolution) {
177             if (result.status.isSuccess) {
178                 Log.d("P2PService", "Conexión establecida con $endpointId")
179                 connectedEndpoints.add(endpointId)
180                 // reiniciamos backoff | si hay una conexión establecida,
                     ↪  preferimos discovery estable
181                 currentRediscoveryDelay = rediscoveryIntervalInitial
182                 //Avisar al modulo de la conexion
183                 onConnected?.invoke(endpointId)
184             } else {
185                 Log.e("P2PService", "Conexión fallida con $endpointId")
186             }
187         }
```

```
188
189         override fun onDisconnected(endpointId: String) {
190             Log.d("P2PService", "Desconectado de £endpointId")
191             connectedEndpoints.remove(endpointId)
192             restartDiscovery(immediate = true) // reintentar de inmediato
193         }
194     }
195
196     // Recepción de datos
197     private val payloadCallback = object : PayloadCallback() {
198         override fun onPayloadReceived(endpointId: String, payload: Payload) {
199             payload.asBytes()?.let {
200                 val message = String(it)
201                 Log.d("P2PService", "Datos recibidos: £message")
202                 onDataReceived?.invoke(message)
203             }
204         }
205
206         override fun onPayloadTransferUpdate(endpointId: String, update:
          ↪  PayloadTransferUpdate) {
207             // Mandar informacion pesada
208         }
209     }
210
211 }
```

E.2. Implementación en PASEO (Java)

En la Figura 4.2 del cuerpo principal se muestra un diagrama de secuencia que ilustra este flujo, mientras que el código recoge la implementación íntegra del servicio. El código del servicio se puede consultar a continuación

```
1  package es.unizar.eina.backend.p2p;
2
3  import android.content.Context;
4  import android.os.Handler;
5  import android.os.Looper;
6  import android.util.Log;
7
8  import com.google.android.gms.nearby.Nearby;
9  import com.google.android.gms.nearby.connection.AdvertisingOptions;
10 import com.google.android.gms.nearby.connection.ConnectionInfo;
11 import com.google.android.gms.nearby.connection.ConnectionLifecycleCallback;
12 import com.google.android.gms.nearby.connection.ConnectionResolution;
13 import com.google.android.gms.nearby.connection.ConnectionsClient;
14 import com.google.android.gms.nearby.connection.DiscoveredEndpointInfo;
15 import com.google.android.gms.nearby.connection.DiscoveryOptions;
16 import com.google.android.gms.nearby.connection.EndpointDiscoveryCallback;
17 import com.google.android.gms.nearby.connection.Payload;
18 import com.google.android.gms.nearby.connection.PayloadCallback;
19 import com.google.android.gms.nearby.connection.PayloadTransferUpdate;
20 import com.google.android.gms.nearby.connection.Strategy;
21
22 import java.util.HashSet;
```

```
23 import java.util.Set;
24
25 public class P2PService {
26
27     private final Handler handler = new Handler(Looper.getMainLooper());
28     private final ConnectionsClient connectionsClient;
29     private final Strategy strategy = Strategy.P2P_CLUSTER;
30     private final Set<String> connectedEndpoints = new HashSet<>();
31
32     // Estado interno
33     private boolean running = false;
34
35     // Callbacks externos
36     private OnDataReceivedListener onDataReceived;
37     private OnConnectedListener onConnected;
38
39     // Valores configurables
40     private final long rediscoveryIntervalInitial = 30_000L;   // 30s
41     private final long rediscoveryMaxDelay = 5 * 60_000L;      // 5min
42     private long currentRediscoveryDelay = rediscoveryIntervalInitial;
43
44     // Guardar valores iniciales para relanzar
45     private String storedServiceName;
46     private String storedUserId;
47
48     public interface OnDataReceivedListener {
49         void onDataReceived(String data);
50     }
51
52     public interface OnConnectedListener {
53         void onConnected(String endpointId);
54     }
55
56     public P2PService(Context context) {
57         this.connectionsClient = Nearby.getConnectionsClient(context);
58     }
59
60     // Runnable que relanza discovery periódicamente
61     private final Runnable rediscoveryRunnable = new Runnable() {
62         @Override
63         public void run() {
64             if (running) {
65                 Log.d("P2PService", "Watchdog: forzando discovery (delay=" +
                    ➝  currentRediscoveryDelay + ")");
66                 restartDiscovery(false);
67                 handler.postDelayed(this, currentRediscoveryDelay);
68             }
69         }
70     };
71
72     // ====================
73     // API pública
74     // ====================
75
76     public void start(String serviceName, String userId) {
77         storedServiceName = serviceName;
78         storedUserId = userId;
79
```

```
80          startAdvertising(serviceName, userId);
81          startDiscovery(serviceName);
82
83          running = true;
84          currentRediscoveryDelay = rediscoveryIntervalInitial;
85
86          handler.postDelayed(rediscoveryRunnable, currentRediscoveryDelay);
87      }
88
89      public void stop() {
90          try {
91              connectionsClient.stopAdvertising();
92              connectionsClient.stopDiscovery();
93              connectionsClient.stopAllEndpoints();
94          } catch (Exception e) {
95              Log.w("P2PService", "Error al parar Nearby: " + e.getMessage());
96          }
97
98          connectedEndpoints.clear();
99          running = false;
100         Log.d("P2PService", "Servicio parado");
101
102         handler.removeCallbacks(rediscoveryRunnable);
103         currentRediscoveryDelay = rediscoveryIntervalInitial;
104     }
105
106     public boolean isStopped() {
107         return !running;
108     }
109
110     public void sendData(String data) {
111         Payload payload = Payload.fromBytes(data.getBytes());
112         for (String endpointId : connectedEndpoints) {
113             connectionsClient.sendPayload(endpointId, payload);
114         }
115     }
116
117     public void setOnDataReceived(OnDataReceivedListener listener) {
118         this.onDataReceived = listener;
119     }
120
121     public void setOnConnected(OnConnectedListener listener) {
122         this.onConnected = listener;
123     }
124
125     // ====================
126     // Nearby API
127     // ====================
128
129     private void startAdvertising(String serviceName, String userId) {
130         AdvertisingOptions options = new
             ↪  AdvertisingOptions.Builder().setStrategy(strategy).build();
131         connectionsClient.startAdvertising(
132                 userId,
133                 serviceName,
134                 connectionLifecycleCallback,
135                 options
136         ).addOnSuccessListener(unused ->
```

```
137                 Log.d("P2PService", "Advertising iniciado")
138         ).addOnFailureListener(e ->
139                 Log.e("P2PService", "Error en advertising: " + e.getMessage())
140         );
141     }
142 
143     private void startDiscovery(String serviceName) {
144         DiscoveryOptions options = new
          → DiscoveryOptions.Builder().setStrategy(strategy).build();
145         connectionsClient.startDiscovery(
146                 serviceName,
147                 endpointDiscoveryCallback,
148                 options
149         ).addOnSuccessListener(unused -> {
150             Log.d("P2PService", "Discovery iniciado");
151             currentRediscoveryDelay = rediscoveryIntervalInitial;
152         }).addOnFailureListener(e ->
153                 Log.e("P2PService", "Error en discovery: " + e.getMessage())
154         );
155     }
156 
157     private void restartDiscovery(boolean immediate) {
158         try {
159             connectionsClient.stopDiscovery();
160         } catch (Exception e) {
161             Log.w("P2PService", "stopDiscovery fallo: " + e.getMessage());
162         }
163 
164         long delay = immediate ? 0L : currentRediscoveryDelay;
165         handler.postDelayed(() -> {
166             if (storedServiceName != null) {
167                 startDiscovery(storedServiceName);
168             } else {
169                 Log.w("P2PService", "No storedServiceName; no se puede relanzar
                  → discovery");
170             }
171         }, delay);
172 
173         currentRediscoveryDelay = Math.min(currentRediscoveryDelay * 2,
          → rediscoveryMaxDelay);
174     }
175 
176     // ====================
177     // Callbacks
178     // ====================
179 
180     private final EndpointDiscoveryCallback endpointDiscoveryCallback = new
      → EndpointDiscoveryCallback() {
181         @Override
182         public void onEndpointFound(String endpointId, DiscoveredEndpointInfo
          → info) {
183             Log.d("P2PService", "Dispositivo encontrado: " + endpointId + " (" +
              → info.getEndpointName() + ")");
184             connectionsClient.requestConnection("P2PDevice", endpointId,
              → connectionLifecycleCallback);
185         }
186 
187         @Override
```

```
188         public void onEndpointLost(String endpointId) {
189             Log.d("P2PService", "Dispositivo perdido: " + endpointId);
190             connectedEndpoints.remove(endpointId);
191             restartDiscovery(false);
192         }
193     };
194 
195     private final ConnectionLifecycleCallback connectionLifecycleCallback = new
        → ConnectionLifecycleCallback() {
196         @Override
197         public void onConnectionInitiated(String endpointId, ConnectionInfo
            → connectionInfo) {
198             Log.d("P2PService", "Conexión iniciada con " + endpointId);
199             connectionsClient.acceptConnection(endpointId, payloadCallback);
200         }
201 
202         @Override
203         public void onConnectionResult(String endpointId, ConnectionResolution
            → result) {
204             if (result.getStatus().isSuccess()) {
205                 Log.d("P2PService", "Conexión establecida con " + endpointId);
206                 connectedEndpoints.add(endpointId);
207                 currentRediscoveryDelay = rediscoveryIntervalInitial;
208                 if (onConnected != null) {
209                     onConnected.onConnected(endpointId);
210                 }
211             } else {
212                 Log.e("P2PService", "Conexión fallida con " + endpointId);
213             }
214         }
215 
216         @Override
217         public void onDisconnected(String endpointId) {
218             Log.d("P2PService", "Desconectado de " + endpointId);
219             connectedEndpoints.remove(endpointId);
220             restartDiscovery(true);
221         }
222     };
223 
224     private final PayloadCallback payloadCallback = new PayloadCallback() {
225         @Override
226         public void onPayloadReceived(String endpointId, Payload payload) {
227             if (payload.asBytes() != null) {
228                 String message = new String(payload.asBytes());
229                 Log.d("P2PService", "Datos recibidos: " + message);
230                 if (onDataReceived != null) {
231                     onDataReceived.onDataReceived(message);
232                 }
233             }
234         }
235 
236         @Override
237         public void onPayloadTransferUpdate(String endpointId,
            → PayloadTransferUpdate update) {
238             // A futuro, manejo de transferencias mas grandes (archivos, etc.)
239         }
240     };
241 }
```

242

Anexos F

Manual de usuario para despliegue y prueba de R-Rules y PASEO

Este anexo recopila las instrucciones necesarias para la instalación, configuración y ejecución de los dos sistemas desarrollados en el marco de este trabajo: *R-Rules* y *PASEO*. Ambos prototipos son aplicaciones independientes, con arquitecturas, entornos de desarrollo y procedimientos de despliegue propios.

En la Sección F.1 se describe el proceso completo para poner en funcionamiento el sistema *R-Rules*, que incluye la configuración de la base de datos, la ejecución del motor de reglas y el despliegue de la aplicación móvil asociada. Por su parte, la Sección F.2 detalla los pasos necesarios para compilar y ejecutar la aplicación *PASEO* en un entorno Android, junto con las configuraciones mínimas requeridas para su correcta ejecución.

El objetivo de este anexo es proporcionar una referencia práctica que permita reproducir y evaluar ambos prototipos de forma autónoma, garantizando la trazabilidad de los resultados presentados en los capítulos experimentales.

F.1. R-Rules

Esta sección describe de manera detallada el procedimiento completo para el despliegue y la ejecución del prototipo *R-Rules*. Se incluyen todas las fases necesarias, desde la preparación de la base de datos y la configuración del entorno de ejecución del motor (*Environment Manager, EM*), hasta la puesta en marcha de la aplicación móvil asociada.

El propósito de esta guía es proporcionar un manual práctico que permita a cualquier usuario reproducir el sistema en su totalidad, garantizando tanto la correcta instalación como la validación del funcionamiento del prototipo. La sección incluye instrucciones paso a paso, configuraciones específicas y comandos necesarios para asegurar un despliegue exitoso del entorno.

Además del procedimiento de instalación, esta sección incorpora un recorrido guiado completo de la aplicación, con capturas de pantalla y explicaciones detalladas de cada funcionalidad, incluyendo la gestión de recomendaciones, la creación de reglas de activación (*Triggering Rules*), conjuntos de exclusión (*Exclusion Sets*) y reglas de contexto (*Context Rules*). De este modo, se proporciona una referencia completa que facilita tanto la utilización del sistema como la realización de pruebas adicionales.

F.1.1. Puesta en marcha del prototipo

En esta sección se presentan los pasos necesarios para poner en marcha el prototipo *R-Rules*. Se describen de manera ordenada los elementos fundamentales para asegurar que el sistema funcione correctamente y que la aplicación móvil pueda interactuar con el *Environment Manager* (EM).

En particular, se detallan los siguientes aspectos:

- Requisitos previos para la instalación y configuración del entorno.
- Despliegue de la base de datos, asegurando que los datos necesarios estén disponibles.
- Despliegue del *Environment Manager (EM)*, responsable de la gestión de reglas y la comunicación con la aplicación móvil.
- Despliegue de la aplicación móvil, garantizando que se pueda ejecutar y conectar correctamente con el EM.

Esta introducción proporciona una visión general de los componentes que se deben iniciar y configurar antes de proceder con la utilización del sistema.

Requisitos previos

Antes de comenzar, se deben instalar las siguientes herramientas:

- **Docker Compose** (versión probada: v2.34.0)
  `https://docs.docker.com/compose/install/`
- **IntelliJ IDEA Ultimate 2025.1.2**
  `https://www.jetbrains.com/idea/`
- **Android Studio** (API Level 35)
  `https://developer.android.com/studio`

- **Node.js** (v22.14.0) y **npm** (v10.9.2)
  `https://nodejs.org/`
- **React Native CLI** (v18.0.0)
  `https://reactnative.dev/docs/environment-setup`
- **Java Development Kit (JDK)**
  Se emplean las versiones:
  - Java 11.0.22 para el EM.
  - Java 17.0.14 para la aplicación Android.

1. Despliegue de la base de datos

1. Acceder al directorio `EM/database`.
2. Ejecutar el siguiente comando:

   ```
   docker-compose up
   ```

   Esto lanzará un contenedor con una instancia de **PostgreSQL** preconfigurada.

Configuración de conexión: La base de datos del sistema puede inspeccionarse o administrarse desde herramientas externas como **DataGrip**, **DBeaver** o **pgAdmin**. Para ello, pueden emplearse los siguientes parámetros de conexión:

- Motor: **PostgreSQL**
- Servidor: `db` (o `localhost` si se accede desde el host)
- Puerto: `5432`
- Usuario: `adminCARS`
- Contraseña: `CARSdb123`
- Base de datos: `CARSdb`

Acceso por terminal: Es posible conectarse directamente por terminal ejecutando:

```
psql postgresql://adminCARS@localhost:5432/CARSdb
# password: CARSdb123
```

Inicialización de datos:

1. Ejecutar el script `createTables.sql` para crear las tablas necesarias.
2. En el directorio `EM/data/scripts`, ejecutar:

   ```
   ./execute.sh
   ```

   Este script poblará la base de datos con datos de prueba.

Creación de usuario de prueba:

- Correo: `test@gmail.com`
- Contraseña: `1234test5678`

Nota: La tabla correspondiente se denomina literalmente `"user"` (con comillas dobles), ya que la palabra `user` es una palabra reservada en SQL.

2. Ejecución del motor EM

El *Environment Manager (EM)* actúa como núcleo de reglas y motor de inferencia del sistema. Para ejecutarlo:

1. Abrir el proyecto `EM` en **IntelliJ IDEA 2025.1.2**.
2. Seleccionar el SDK de Java **11.0.22**.
3. Compilar y ejecutar el módulo principal del EM.

Una vez en ejecución, el servicio expone una API REST documentada mediante **Swagger** y **OpenAPI 3.0** en la siguiente dirección:

`http://localhost:8080/swagger-ui.html`

3. Ejecución de la aplicación móvil

Configuración del entorno:

- Android Studio (API Level 35)
- Node.js 22.14.0 y npm 10.9.2
- React Native 0.79.1
- Java 17.0.14

Ejecución en emulador Android La aplicación móvil puede ejecutarse en un emulador Android, lo que permite probar su funcionamiento sin necesidad de un dispositivo físico. Esto facilita la verificación de la interfaz de usuario, la interacción con el *Environment Manager* (EM) y el comportamiento general del prototipo en condiciones controladas.

Pasos para el despliegue:

1. Abrir el proyecto `MobileApp` en **Android Studio**.
2. Sincronizar dependencias de Gradle.
3. Abrir un terminal en el directorio del proyecto y ejecutar:
   ```
   npm install
   ```
4. Lanzar el servidor de desarrollo:
   ```
   npm run start
   ```
5. Iniciar un emulador Android con API Level 35 y, desde otra terminal, ejecutar:
   ```
   adb reverse tcp:8081 tcp:8081
   adb reverse tcp:8080 tcp:8080
   npm run android
   ```

Esto permitirá ejecutar la aplicación móvil conectada al servidor local del EM, validando el funcionamiento completo del prototipo.

Ejecución en un dispositivo Android real La aplicación también puede desplegarse en un dispositivo Android físico, lo que permite evaluar su comportamiento bajo condiciones reales de uso, incluyendo el acceso a sensores y la comunicación con servicios externos.

Requisitos previos:

- Dispositivo Android con versión igual o superior a Android 13.
- Opciones de desarrollador activadas.
- **Depuración USB** habilitada.
- Cable USB compatible.

Pasos para el despliegue:

1. Conectar el dispositivo Android al ordenador mediante USB.
2. Verificar que ADB lo detecta correctamente:

   ```
   adb devices
   ```

3. En el directorio del proyecto, instalar dependencias:

   ```
   npm install
   ```

4. Iniciar el servidor de desarrollo:

   ```
   npm run start
   ```

5. Si el *Environment Manager* se ejecuta en la misma máquina, redirigir puertos:

   ```
   adb reverse tcp:8081 tcp:8081
   adb reverse tcp:8080 tcp:8080
   ```

6. Desplegar la aplicación en el dispositivo:

   ```
   npm run android
   ```

Notas:

- Se recomienda desactivar restricciones de batería para evitar que el dispositivo limite la ejecución en segundo plano durante las pruebas.

F.1.2. Recorrido guiado de la aplicación

En esta sección se presenta un recorrido guiado por la aplicación *R-Rules*, con el objetivo de familiarizar al usuario con su funcionamiento y principales características. A través de diferentes pantallas se mostrarán las opciones disponibles, cómo interactuar con ellas y los pasos necesarios para realizar las operaciones más habituales. Este recorrido guiado permitirá comprender de forma práctica el uso de la aplicación y servirá como referencia para los apartados posteriores del manual.

Pantallas principales de navegación

La aplicación cuenta con tres vistas principales: *Home*, *Saved* e *Historic*, que conforman el flujo básico de consulta y gestión de recomendaciones por parte del usuario.

Al iniciar la aplicación, el usuario accede directamente a la pantalla *Home* (Figura F.1), que constituye el punto de entrada principal. Desde esta vista se muestran las recomendaciones actuales generadas por el sistema.

En la parte inferior se encuentra la barra de navegación principal, que permite desplazarse entre las distintas secciones de forma sencilla:

– **Home**: listado de recomendaciones activas.

– **Saved**: recomendaciones marcadas como favoritas.

– **Historic**: historial de recomendaciones visualizadas anteriormente.

En la sección *Saved*, se encuentran todas las recomendaciones que el usuario haya marcado como favoritas. En esta vista también se permite interactuar con cada elemento mediante gestos, tal y como se muestra en la Figura F.2.

Finalmente, la pantalla *Historic* (Figura F.3) muestra el historial de recomendaciones visualizadas anteriormente. Esta vista no contempla acciones mediante gestos y se limita a la consulta del registro histórico.

Home

La Chalana (Plaza Españ,a) #Restaurants

Radio Rooftop #Restaurants

Zombie Bar #Restaurants

El Cocinillas #Restaurants

The One #Restaurants

¡,Tu! Pasta #Restaurants

La Buga del Lobo #Restaurants

Toma Jamó,n (Vergara) #Restaurants

[A]Brasa #Restaurants

(a) Pantalla Home

Home

Chanel #Shops

Tous (Goya) #Shops

Energí,a Positiva Siglo XXI #Shops

Patch #Shops

Castellana 200 #Shops

Arte granda #Shops

Mercado de Barceló, #Shops

(b) Opción de favorito

Home

Shops

Tous (Goya) #Shops

Energí,a Positiva Siglo XXI #Shops

Patch #Shops

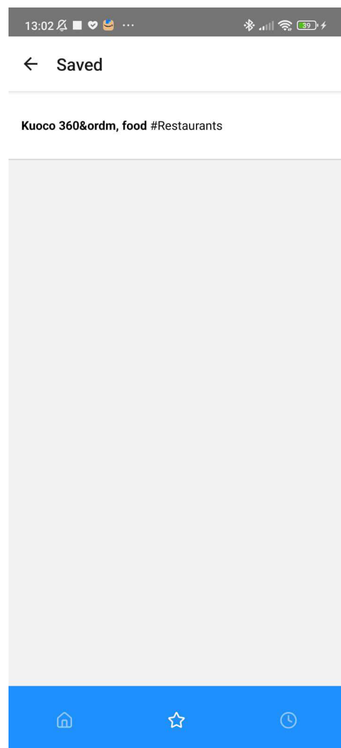
Castellana 200 #Shops

Arte granda #Shops

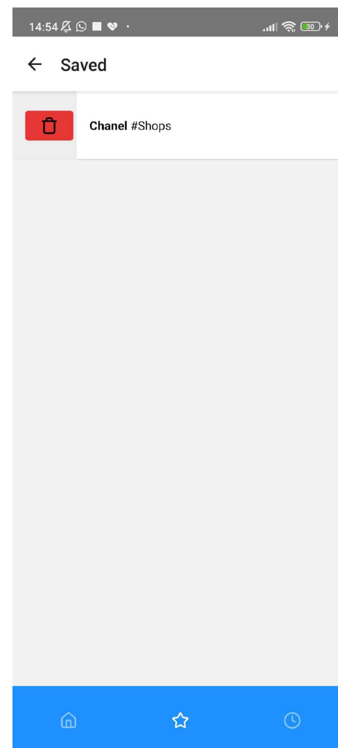
Mercado de Barceló, #Shops

(c) Opción de eliminar

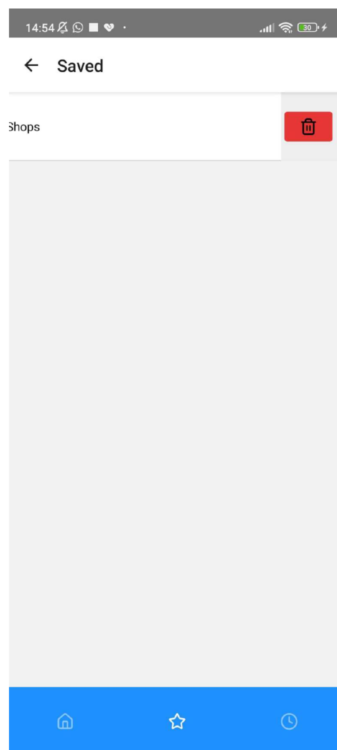
Figura F.1: Interacción básica con las recomendaciones en la pantalla principal.



(a) Pantalla Saved



(b) Quitar de favoritos



(c) Eliminar recomendación

Figura F.2: Gestión de recomendaciones marcadas como favoritas.

Figura F.3: Pantalla Historic.

Menú lateral de navegación

La aplicación dispone de un panel lateral accesible desde cualquier pantalla mediante un gesto de deslizamiento desde el borde izquierdo hacia la derecha. Este menú permite acceder rápidamente a las secciones principales de la aplicación sin necesidad de volver a la pantalla inicial.

El menú ofrece acceso directo a:

- **Home**: vista principal de recomendaciones.
- **Profile**: información asociada al usuario.
- **Settings**: opciones generales de configuración.
- **Recommendation Triggering Rules**: gestión y creación de reglas de activación.
- **Recommendations Exclusions and Priorities**: definición de conjuntos de exclusión y prioridades.
- **Context Rules**: creación y gestión de reglas de contexto.

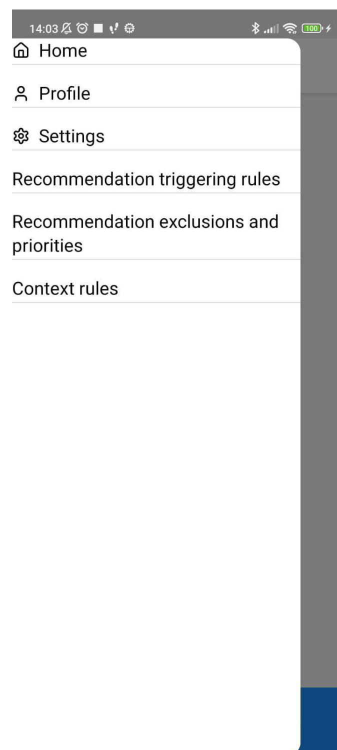


Figura F.4: Menú lateral de navegación.

Pantalla de perfil

La Figura F.5 corresponde a la pantalla *Profile*, donde el usuario puede consultar los parámetros clave relacionados con la interacción entre la aplicación y el *Environment Manager* (EM).

En esta vista se muestran los datos de contexto que la aplicación puede compartir con el EM, tales como ubicación, información temporal u otros atributos relevantes para la generación de recomendaciones. Además, se indican dos configuraciones importantes del sistema:

- **Intervalo de envío de información**: especifica cada cuántos minutos la aplicación envía los datos de contexto al EM.

- **Radio máximo de descubrimiento**: distancia máxima (en kilómetros) dentro de la cual la aplicación buscará y detectará EM disponibles, determinando el alcance de la señal de descubrimiento.

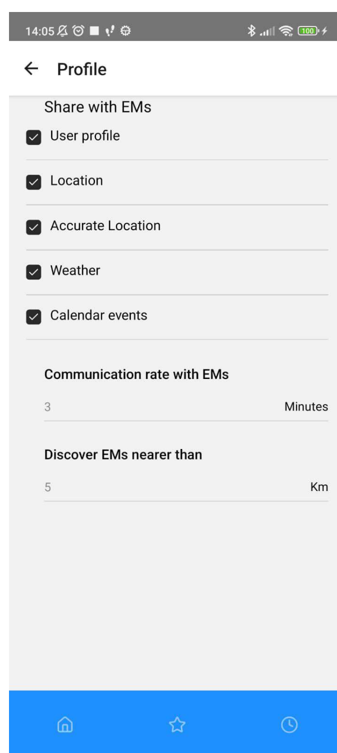


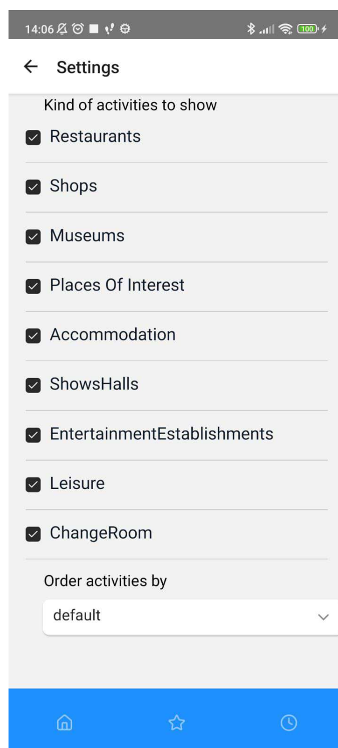
Figura F.5: Pantalla Profile.

Pantalla de configuración

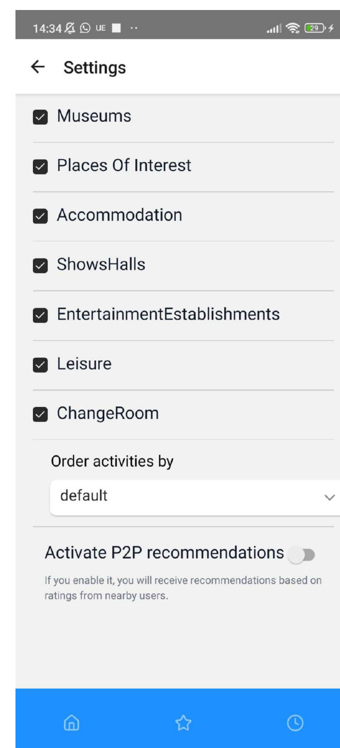
La pantalla *Settings* (Figura F.6) permite al usuario configurar las preferencias generales de la aplicación relacionadas con la visualización de recomendaciones.

En esta sección es posible seleccionar qué tipos de recomendaciones desea recibir el usuario (por ejemplo, restaurantes, tiendas, museos, entre otros), permitiendo personalizar el contenido mostrado en la pantalla *Home*. Asimismo, la pantalla incluye opciones para definir el criterio de ordenación del listado de recomendaciones, ya sea mostrando primero las más recientes, priorizando las más relevantes o aplicando otros mecanismos de ordenación según las preferencias del usuario.

Al deslizar hacia abajo se accede a un ajuste adicional que permite habilitar la recepción de recomendaciones basadas en valoraciones de otros usuarios cercanos mediante comunicación *peer-to-peer*. Al activar este interruptor, el sistema comenzará a recibir recomendaciones calculadas a partir del rating de otros usuarios en el entorno cercano. Esta opción por defecto está desactivada por medidas de seguridad.



(a) Opciones principales



(b) Opciones principales parte de abajo

Figura F.6: Pantalla Settings de la aplicación.

Recommendation Triggering Rules

La sección *Recommendation Triggering Rules* permite al usuario gestionar las reglas que activan la generación de recomendaciones. Desde este módulo es posible consultar las reglas existentes, crear nuevas, acceder a su vista detallada o eliminarlas mediante gestos. A continuación se describen las diferentes pantallas que forman parte del flujo de uso.

Al acceder a este apartado, la primera pantalla que se muestra es el listado de *Triggering Rules* creadas, como se observa en la Figura F.7. Desde esta vista, el usuario puede:

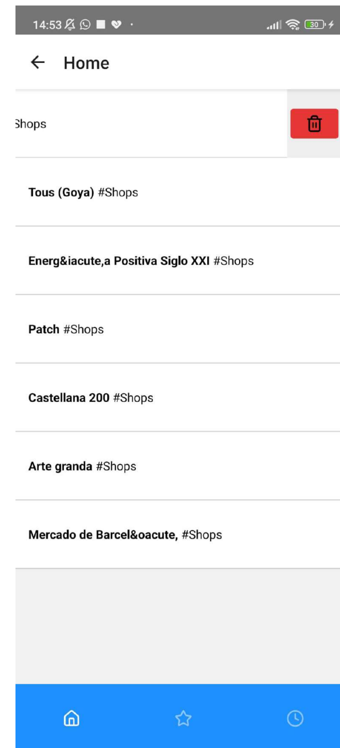
- Visualizar todas las reglas creadas previamente.
- Deslizar una regla hacia la izquierda para mostrar la opción de borrado.
- Pulsar sobre una regla para acceder a su vista detallada.
- Activar o desactivar una regla mediante el interruptor disponible en cada entrada.

Tras confirmar el borrado de una regla, la aplicación muestra un mensaje indicando que la operación se ha completado correctamente, como se aprecia en la Figura F.8.

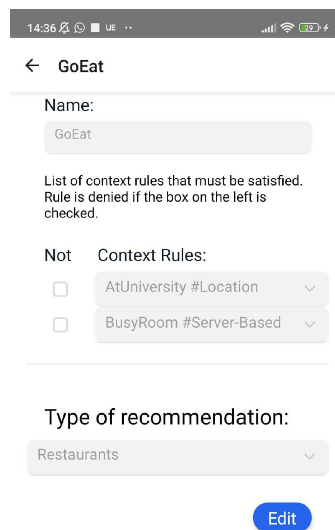
El proceso de creación se compone de varias pantallas que guían al usuario paso a paso. En primer lugar, se selecciona el tipo de recomendación que se generará cuando la regla se active, como se muestra en la Figura F.9.

Una vez seleccionados todos los parámetros necesarios, la aplicación confirma que la nueva regla se ha creado correctamente, como puede verse en la Figura F.10.

(a) Listado



(b) Borrar



(c) Detalle

Figura F.7: Gestión de Triggering Rules.

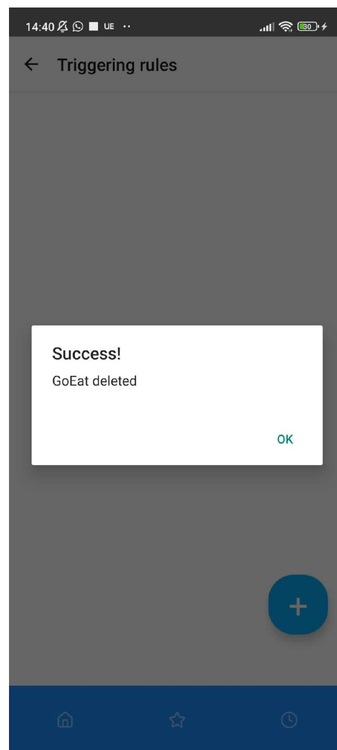
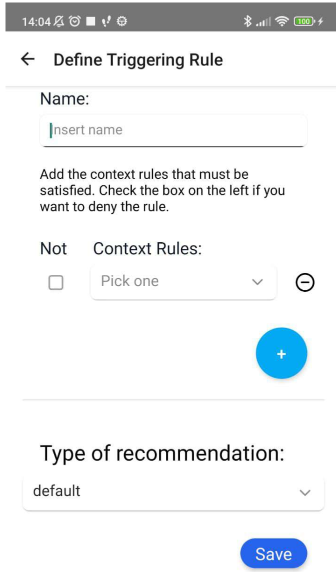
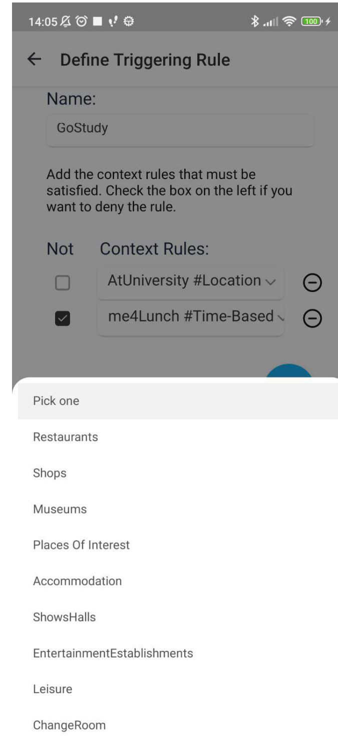


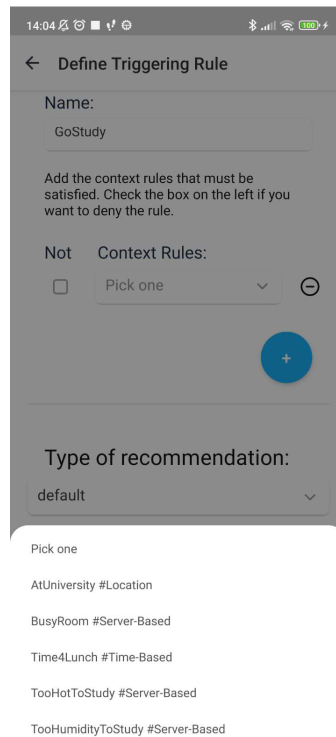
Figura F.8: Confirmación de eliminación de la Triggering Rule



(a) Configuración final



(b) Tipo



(c) Contexto

Figura F.9: Creación de Triggering Rule (parte 1).

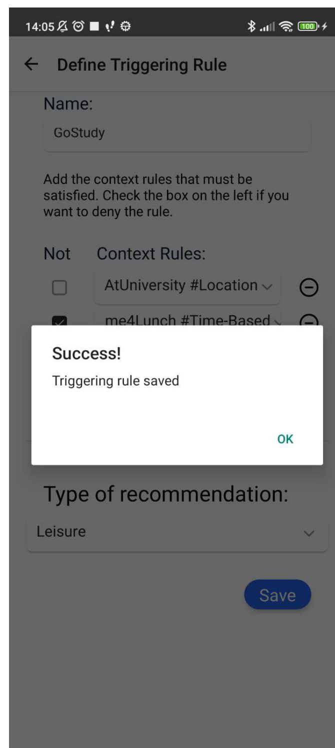


Figura F.10: Confirmación de creación exitosa de la Triggering Rule

Gestión de Conjuntos de Exclusión (Exclusion Sets)

Los *exclusion sets* permiten definir grupos de reglas o condiciones que no pueden cumplirse simultáneamente, evitando que determinadas recomendaciones o acciones se activen a la vez. Esta funcionalidad mantiene un comportamiento coherente dentro del sistema y previene conflictos entre reglas de contexto o *triggering rules*.

La pantalla principal muestra todos los conjuntos de exclusión actualmente definidos (Figura F.11). Desde aquí, el usuario puede consultar, crear o eliminar *exclusion sets*. Además, al lado de cada conjunto hay flechas hacia arriba y hacia abajo que permiten cambiar su posición en la lista; cuanto más arriba se encuentre un conjunto, mayor prioridad tendrá frente a los demás.

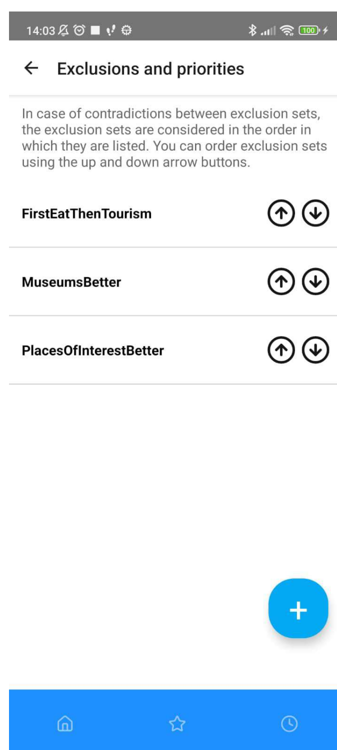


Figura F.11: Pantalla principal de Exclusion Sets.

Cuando el usuario selecciona la opción de crear un nuevo *exclusion set*, aparece la pantalla inicial donde puede introducir el nombre del conjunto y seleccionar las reglas que no pueden cumplirse simultáneamente (Figura F.12).

Una vez definido el conjunto y pulsado el botón de guardar, la aplicación muestra un mensaje de confirmación indicando que el *exclusion set* se ha creado correctamente (Figura F.13).

Al seleccionar uno de los conjuntos definidos en la lista principal, el usuario accede a una vista detallada donde puede revisar su contenido y las reglas que lo componen (Figura F.14).

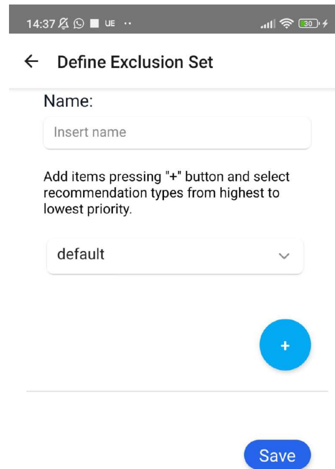


Figura F.12: Creación de un nuevo Exclusion Set.

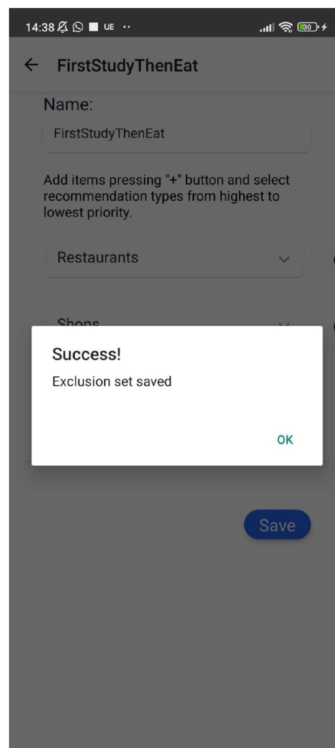


Figura F.13: Confirmación de creación de Exclusion Set.

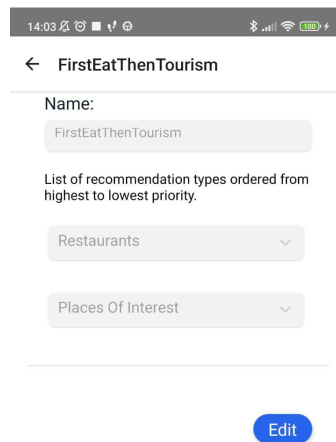
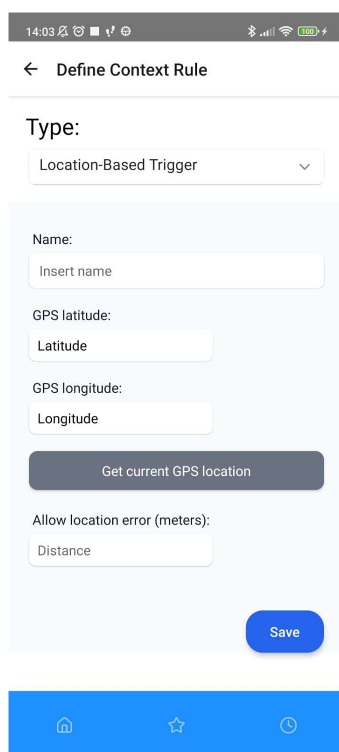


Figura F.14: Detalle de un Exclusion Set.

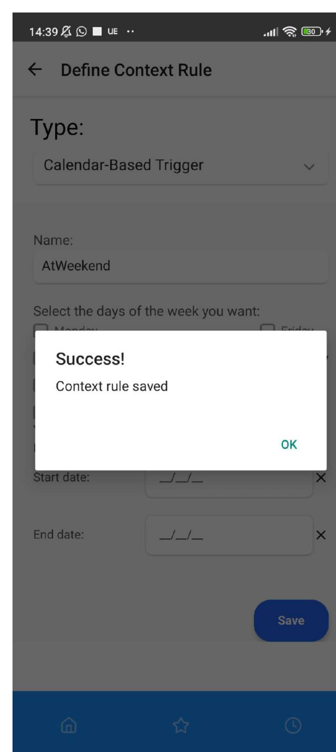
Gestión de Reglas de Contexto (Context Rules)

El sistema permite crear reglas contextuales basadas en ubicación, tiempo u otras señales de interés, las cuales pueden condicionar la activación de recomendaciones específicas. Esta sección describe el flujo de creación, confirmación y visualización de dichas reglas.

Para crear una nueva regla contextual, el usuario selecciona los parámetros deseados, como la localización, hora o condiciones adicionales. La Figura F.15 muestra un ejemplo de creación de una regla basada en localización y la confirmación posterior de su creación.



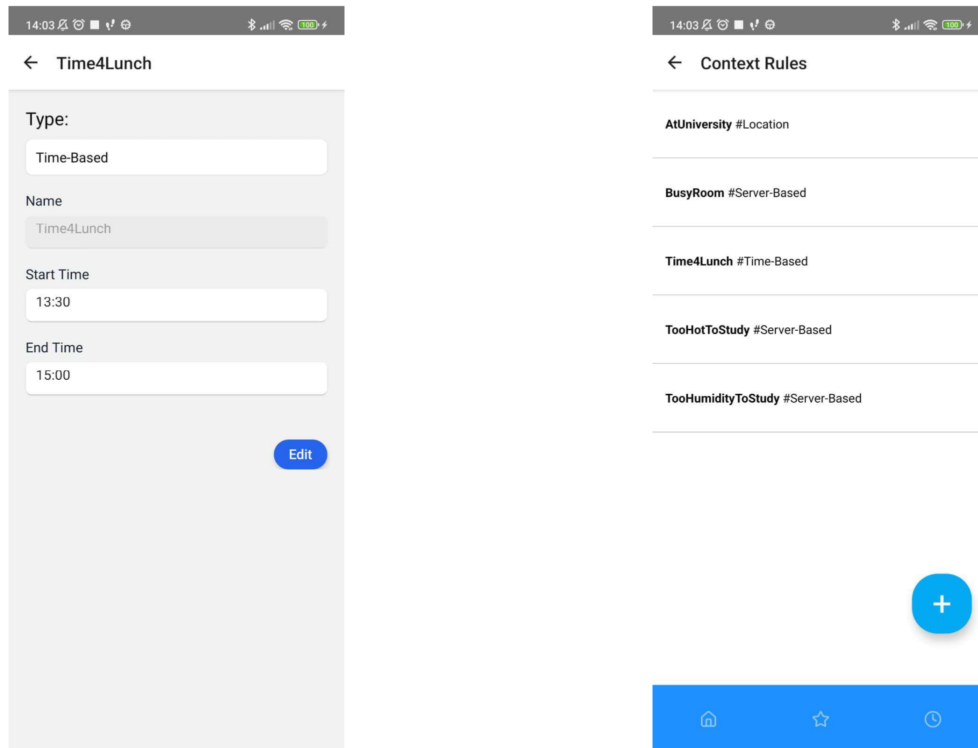
(a) Creación de regla



(b) Confirmación de guardado

Figura F.15: Creación de regla contextual.

En cualquier momento, el usuario puede consultar las reglas de contexto existentes. La pantalla principal de visualización de reglas permite acceder a la información detallada de cada regla y al menú general de gestión de reglas (Figura F.16).



(a) Detalle de regla (b) Menú de reglas

Figura F.16: Visualización de reglas de contexto.

F.1.3. Verificación del funcionamiento de la aplicación

Antes de probar la aplicación móvil, es importante comprobar que el *Environment Manager* (EM) está funcionando correctamente. Para ello, el usuario puede acceder a la interfaz Swagger disponible en: `http://localhost:8080/swagger-ui.html`. Si el EM está activo, se mostrará la documentación generada por Swagger, indicando que el servicio responde correctamente.

Además, si se lanza el EM desde IntelliJ (véase Sección F.1.1), al conectarse la aplicación móvil se pueden observar los siguientes mensajes en la consola del EM:

- `TOKENFORMATISVALID true`
- `TOKENISVALID TRUE`

Estos mensajes confirman que la aplicación móvil se ha conectado correctamente al EM y que los tokens de autenticación son válidos.

Una vez verificado el EM, el usuario puede comprobar el funcionamiento de la aplicación móvil realizando los siguientes pasos:

- Crear una regla de contexto simple (por ejemplo, basada en localización).
- Asociarla a una *Triggering Rule*.

- Revisar que la aplicación muestre una recomendación coherente en la pantalla *Home*.

F.1.4. Notas adicionales

- No se requiere configuración adicional ni conexión a Internet para las pruebas básicas de R-Rules.
- Si se desea utilizar las funcionalidades de comunicación P2P, deben emplearse al menos un dispositivo Android físico con servicios de Google Play activos (requisito de la API *Nearby Connections*).

F.2. Despliegue y ejecución de PASEO

El sistema *PASEO* está implementado íntegramente como una aplicación Android nativa. A diferencia de *R-Rules*, no requiere componentes externos ni servicios de backend para su funcionamiento básico, ya que integra en el propio dispositivo tanto la lógica de recomendación como la gestión de datos locales.

Esta sección describe el procedimiento necesario para compilar y ejecutar el prototipo, así como los requisitos mínimos del entorno de desarrollo. Se detalla el proceso de sincronización del proyecto, la configuración de puertos para pruebas locales y la ejecución de la aplicación en emuladores o dispositivos físicos. Además, se incluyen consideraciones específicas relacionadas con las pruebas de comunicación *peer-to-peer* (P2P) y el uso de la API *Nearby Connections*, empleada para el intercambio directo de información entre dispositivos.

F.2.1. Puesta en marcha del prototipo

Esta sección describe el proceso necesario para preparar y ejecutar el prototipo *PASEO* en un entorno de desarrollo Android. A diferencia de *R-Rules*, *PASEO* no requiere el despliegue de servicios externos ni de contenedores adicionales, ya que todo el procesamiento se realiza de forma local en el dispositivo. El objetivo es proporcionar una guía práctica que permita ejecutar la aplicación de forma inmediata, ya sea en un emulador o en un dispositivo físico Android, garantizando la reproducción del entorno utilizado durante las pruebas experimentales.

Entorno de desarrollo

Para el desarrollo y ejecución del prototipo se empleó el siguiente entorno:

- **IDE:** Android Studio 4.0.1

- **Android SDK:** Nivel de API 29
- **Versión de Java:** 8.0

Preparación del entorno

Para poner en funcionamiento el proyecto basta con abrir el directorio raíz `PASEO` en Android Studio y seguir los pasos que se detallan a continuación:

1. Sincronizar las dependencias del proyecto mediante la función de Android Studio:

   ```
   Sync Project with Gradle Files
   ```

2. Conectar un dispositivo físico o iniciar un emulador con API 29 o superior.

3. En caso de emplear un backend de pruebas en la máquina local, redirigir el puerto de comunicación hacia el dispositivo Android:

   ```
   adb reverse tcp:8080 tcp:8080
   ```

4. Ejecutar el proyecto desde Android Studio a través de:

   ```
   Run > Run 'app'
   ```

F.2.2. Recorrido guiado de la aplicación

En esta sección se realiza un recorrido guiado por la aplicación *PASEO* con el objetivo de familiarizar al usuario con su interfaz y funcionamiento básico. A través de las capturas de pantalla se mostrarán las pantallas principales de la aplicación, las acciones más habituales (por ejemplo, iniciar sesión, consultar puntos de interés y consultar el historial) y los gestos o controles necesarios para interactuar con cada vista.Este recorrido guiado permitirá comprender de forma práctica el uso de la aplicación y servirá como referencia para los apartados posteriores del

Pantalla de registro de usuario

La Figura F.17 muestra la primera pantalla que se presenta al iniciar la aplicación *PASEO*. Desde esta interfaz, el usuario puede crear una cuenta local introduciendo un nombre de usuario, que será utilizado para identificarlo en las funciones internas de la aplicación.

En el ejemplo ilustrado, se ha introducido el nombre *prueba* con el fin de mostrar el proceso de registro de manera clara y comprensible para el usuario final.

Figura F.17: Pantalla inicial de registro de usuario en PASEO

Pantalla inicial de recomendaciones

Tras completar el registro e iniciar sesión, el usuario accede a la pantalla principal de la aplicación, donde se muestran las recomendaciones generadas por el sistema en función de su ubicación y del contexto disponible.

La Figura F.18 presenta un ejemplo de esta vista inicial, en la que se muestra un punto de interés recomendado cercano al usuario. En este caso, el sistema sugiere un polideportivo, ilustrando el funcionamiento básico del motor de recomendaciones de *PASEO*.

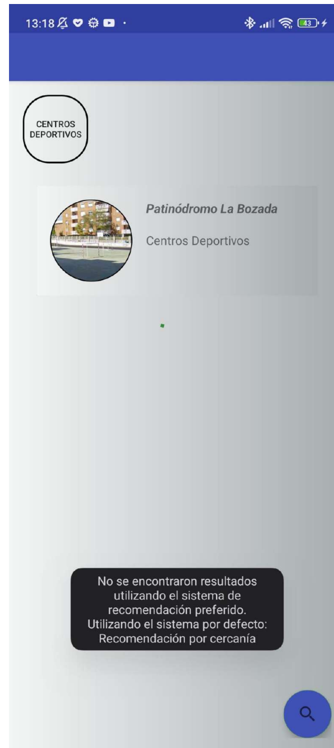


Figura F.18: Pantalla inicial con recomendaciones de puntos de interés cercanos

Pantalla de detalle de Puntos de Interés

La Figura F.19 muestra las pantallas de **detalle de un Punto de Interés (PI)** dentro de la aplicación *PASEO*.

- La subfigura F.19a presenta la información principal del PI, incluyendo su descripción y fotografía, permitiendo al usuario conocer mejor el lugar antes de interactuar con él.
- La subfigura F.19b muestra el mismo PI después de haber sido valorado por el usuario y añadido a la lista de favoritos, indicando claramente cómo se reflejan las acciones del usuario sobre los PIs.

Estas pantallas permiten al usuario consultar en detalle cada lugar recomendado y gestionar sus preferencias de manera intuitiva.

(a) Detalles de PI

(b) PI valorado y añadido a favoritos

Figura F.19: Pantallas de detalle de Puntos de Interés (PIs) en PASEO

Menú principal de navegación

La aplicación cuenta con un menú lateral accesible en cualquier momento deslizando desde el borde izquierdo de la pantalla. Este menú, mostrado en la Figura F.20, permite acceder de forma rápida a las diferentes secciones de *PASEO*.

En la parte superior del menú se incluye el logotipo de la aplicación, que actúa como acceso directo a la pantalla principal de recomendaciones (ver Figura F.18). A continuación, se presentan las distintas opciones disponibles:

- **Ajustes:** configuración general de la aplicación.
- **Valoraciones:** acceso a la sección para puntuar lugares visitados.
- **Favoritos:** listado de puntos marcados por el usuario como destacados.
- **Acerca de:** información general y datos de créditos de la aplicación.

Figura F.20: Menú principal de navegación lateral

Pantalla de ajustes de la aplicación

La Figura F.21 muestra la pantalla de **Ajustes**, accesible desde el menú lateral de navegación. Desde esta sección, el usuario puede configurar diversos parámetros generales relacionados con el funcionamiento de la aplicación *PASEO*.

En particular, se pueden modificar ajustes que afectan al comportamiento del sistema y a la personalización de la experiencia, tales como:

- Preferencias de interacción y visualización.
- Parámetros asociados al funcionamiento interno del sistema.
- Opciones de control o depuración para el envío de información.

Esta pantalla constituye el punto central para adaptar la aplicación a las necesidades específicas de cada usuario, permitiendo ajustar su uso sin necesidad de reiniciar ni reinstalar el sistema.

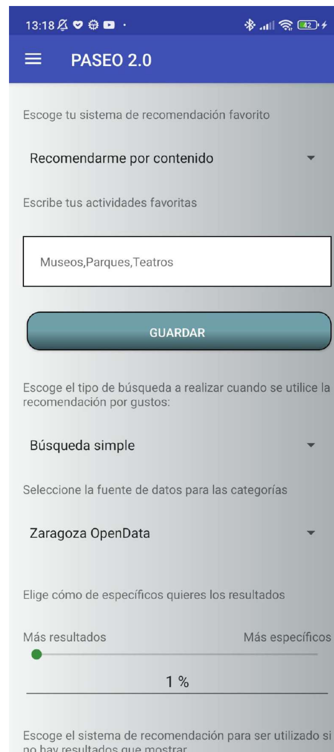


Figura F.21: Pantalla de ajustes de la aplicación

Pantalla de Puntos de Interés valorados

La Figura F.22 muestra la pantalla **Valorados**, accesible desde el menú lateral de la aplicación. En esta sección, el usuario puede consultar todos los Puntos de Interés que ha valorado previamente, permitiendo un seguimiento de sus interacciones con la aplicación.

Para facilitar la comprensión, la figura incluye varios ejemplos de Puntos de Interés con sus valoraciones correspondientes, ilustrando cómo se registran y visualizan las preferencias del usuario en *PASEO*.

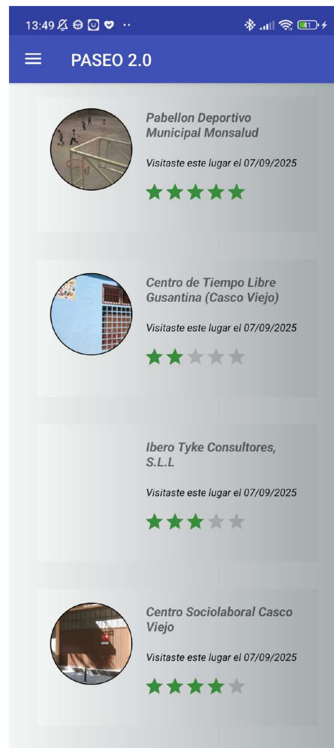


Figura F.22: Pantalla de Puntos de Interés valorados por el usuario

Pantalla informativa: Acerca de PASEO

La Figura F.23 presenta la pantalla **Acerca de PASEO**, accesible desde el menú lateral de la aplicación. En esta sección, el usuario puede consultar información general sobre la aplicación, incluyendo su finalidad, versión, autores y otros datos relevantes que permiten comprender mejor el sistema y su contexto de uso.

Esta pantalla proporciona un recurso útil para que los usuarios obtengan detalles sobre la aplicación y su desarrollo sin afectar la funcionalidad principal de *PASEO*.

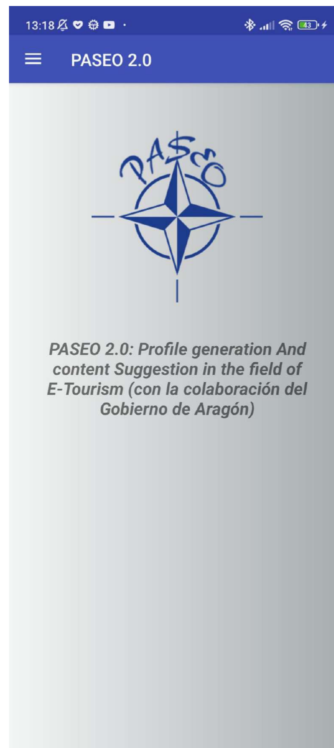


Figura F.23: Pantalla informativa: Acerca de PASEO

F.2.3. Verificación del funcionamiento de la aplicación

Para asegurarse de que el prototipo *PASEO* funciona correctamente, se recomienda realizar las siguientes comprobaciones:

- **Registro e inicio de sesión:** Crear un nuevo usuario y verificar que puede iniciar sesión correctamente. Comprobar que los datos introducidos durante el registro se guardan y se muestran adecuadamente.

- **Visualización de Puntos de Interés (PIs):** Al iniciar sesión, la pantalla principal debe mostrar los PIs recomendados al usuario. Se debe comprobar que cada PI incluye información completa como nombre, descripción, fotografía y ubicación.

- **Gestión de favoritos:** Añadir un PI a favoritos y verificar que aparece en la lista de favoritos. Eliminar un PI de favoritos y comprobar que desaparece correctamente de la lista.

- **Valoración de PIs:** Asignar una valoración a un PI y comprobar que se refleja en la lista de valorados. Revisar que la valoración se mantiene tras cerrar y volver a abrir la aplicación.

- **Navegación mediante el menú lateral:** Comprobar que cada opción del menú lateral funciona correctamente y conduce a la pantalla correspondiente: Ajustes, Valorados, Favoritos y Acerca de. Verificar que pulsar sobre el logo de la aplicación retorna a la pantalla principal de PIs.
- **Ajustes de la aplicación:** Modificar las preferencias en la pantalla de Ajustes y verificar que los cambios se aplican correctamente en la aplicación.

F.2.4. Notas adicionales

- No es necesaria configuración adicional ni conexión a Internet para las funciones básicas de *PASEO*.
- Para activar las capacidades de comunicación *peer-to-peer* basadas en *Nearby Connections*, es necesario utilizar al menos un dispositivo Android físico con servicios de Google Play activos.