Universidad de Cádiz
Escuela Superior de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

SISTEMA DE MONITORIZACIÓN EN TIEMPO REAL DEL TRÁFICO, APARCAMIENTO Y POLUCIÓN EN LAS CIUDADES

AUTOR: MANUEL CANO CRESPO

Cádiz, enero 2023

Universidad de Cádiz
Escuela Superior de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

SISTEMA DE MONITORIZACIÓN EN TIEMPO REAL DEL TRÁFICO, APARCAMIENTO Y POLUCIÓN EN LAS CIUDADES

TUTOR: GUADALUPE ORTIZ BELLOT
COTUTOR: JUAN BOUBETA PUIG
AUTOR: MANUEL CANO CRESPO

Cádiz, enero 2023

Agradecimientos

A mi madre y a mi padre, por haberme puesto todas las facilidades y haberme dado una educación de la que puedo sentirme orgulloso, sin ellos nada de esto hubiera sido posible. Sobre todo a mi madre, sin su insistencia y su apoyo, y sin los mensajes de "¿Dónde estás? Deberías de estar estudiando", seguramente no me encontraría ahora en este punto.

A mi hermano Martín y también a la familia que elegí, mis amigos, porque les debo mucho de lo que soy, por estar cuando uno apenas está, y porque cuando tanto la vida como la carrera se veían como una cuesta arriba, me impulsaron hacia arriba y subieron la cuesta conmigo.

Por supuesto a Guadalupe y a Juan, por su implicación y compromiso en este proyecto. Y a todos los profesores que, aparte de como profesión, entienden la docencia como una vocación y un servicio a la sociedad, y que me han enseñado tanto durante toda mi etapa como estudiante.

También a los estupendos compañeros, algunos ya amigos, con los que he compartido tanto tiempo en la universidad que, huyendo de la espiral de competitividad a la que nos empuja este sistema, no han dudado en ayudarme y permitirme aprender de ellos, y me han hecho mucho más llevaderos todos estos años.

Y, por último y no menos importante, a la educación pública, de la que debemos sentirnos orgullosos, sin que ello signifique ser complacientes y dejar de defenderla desde cualquier terreno para seguir ampliando el derecho a una universidad pública e inclusiva, que no deje a nadie atrás y garantice la igualdad de oportunidades.

Índice general

Índice de figuras

Índice de tablas

1. Visión general del proyecto

En este capítulo se ofrecerá una visión general del proyecto: la motivación que llevó a desarrollarlo, los objetivos que pretende cumplir y el alcance del mismo; así como una visión general de la estructura y los contenidos del presente documento, los estándares que sigue y los acrónimos usados en él, y por último, las definiciones de los conceptos más relevantes del proyecto.

1.1. Motivación

Sin duda el mayor reto que tiene la sociedad por delante es la lucha contra el cambio climático que amenaza con la desaparición de nuestro planeta tal y como lo conocemos. Ya empezamos a ver en nuestro día a día las consecuencias de este: veranos con temperaturas récord, aumento de los fenómenos meteorológicos extremos, sequías históricas, etc.

Consecuentemente, debemos abordar esta problemática desde todos los frentes, y uno de los principales activos en esta lucha deben ser los ayuntamientos de las ciudades. Se debe apostar desde ya por una movilidad y una industria sostenible que hagan de la ciudad una ciudad respirable y habitable para todos.

Cuando hablamos de "Smart cities" o ciudades inteligentes, la reducción de los niveles de contaminación en las ciudades debe ser uno, sino el principal, de los focos donde dirigir los avances tecnológicos. Las tecnologías de la información, y en concreto, lo que atraviesa este proyecto, el procesamiento de eventos complejos, tienen mucho que aportar en el proceso de coordinar, mejorar y automatizar (en la medida de lo posible), la detección y actuación ante situaciones que perjudiquen al medio ambiente.

La Bahía de Cádiz y la Bahía de Algeciras son dos de los diez focos de más contaminación de Andalucía [1], según el informe "La calidad del aire en el Estado español durante 2021" publicado por Ecologistas en Acción[2]. Es evidente por tanto que debemos tomar medidas urgentes para mejorar la calidad del aire que respiramos en nuestras ciudades.

Tomando como referencia la ciudad de Cádiz, en la que vivo, podemos ver claramente que el principal problema, y en el que el ayuntamiento tiene puesto el foco desde hace años, es la movilidad. El gran volumen de vehículos que circula a todas horas por la ciudad emite gases que perjudican al medio ambiente y a nuestra propia salud. Además, la circulación se ve perjudicada por atascos y conductores buscando constantemente aparcamiento, lo que hace que se generen emisiones contaminantes que de no producirse estas situaciones no se generarían.

Con todo esto en mente he desarrollado mi sistema, para monitorizar los niveles de polución del aire y detectar de manera automática situaciones que perjudiquen al medio ambiente y a la salud de la gente, así como ayudar a la búsqueda de aparcamiento y a la detección de atascos, de forma que la actuación para solucionar estas situaciones sea casi inmediata.

1.2. Objetivos

El objetivo principal es diseñar e implementar una arquitectura software que permita tratar una serie de datos relacionados con el tráfico en la ciudad, el aparcamiento y la contaminación atmosférica, mediante un sistema, que lea los datos de una cola de mensajería y los procese en tiempo real con un motor de procesamiento de eventos complejos (CEP).

El sistema deberá ser persistente, almacenando el registro de eventos complejos detectados, de manera que se pueda acceder posteriormente a la información para actuar ante ella.

Además, se desarrollará una aplicación Android que permita visualizar situaciones de interés detectadas de una manera amigable para el usuario. Esta aplicación, aparte de mostrar los detalles de los eventos ocurridos, deberá ubicar geográficamente en un mapa plazas de aparcamientos libres y atascos que hay actualmente en la ciudad.

1.3. Alcance

La aplicación Mule podrá procesar cualquier dato que reciba a través del bróker RabbitMQ siempre que los datos sigan un formato JSON predefinido por el desarrollador de la aplicación. Aunque se utilizarán datos simulados para facilitar la prueba del sistema podrían usarse datos reales siempre que estos utilizasen el formato especificado o se adaptasen previamente a este. Dicho formato vendrá dado por el esquema de eventos que se registrará en el motor de procesamiento de eventos complejos.

Las situaciones de interés detectadas por el sistema se determinarán a través de la definición de una serie de patrones en el lenguaje EPL de Esper (el motor de procesamiento de eventos complejos usado). Si bien será un conjunto limitado de patrones, se podría añadir cualquier otro patrón que se desee usando el lenguaje EPL previamente mencionado para definir patrones cuyos eventos de entrada cumplan los esquemas previamente definidos.

La base de datos almacenará las situaciones de interés que se consultarán mediante una API REST que estará limitada a ofrecer las operaciones necesarias para obtener los datos a mostrar en la app; aunque puede extenderse con operaciones adicionales en caso de que sea necesario.

Por último, la app mostrará la situación relevante para el usuario final en base a las situaciones de interés detectadas por el motor de procesamiento de eventos complejos.

1.4. Visión general del documento

El resto del documento está estructurado de la siguiente forma:

- **Contexto del proyecto**: recoge todo lo que rodea al proyecto; los usuarios a los que está orientado el sistema, el modelo de ciclo de vida elegido, las tecnologías, lenguajes y herramientas utilizadas para desarrollarlo…
- **Planificación**: explica cómo ha sido el avance del proyecto a través del tiempo, describiendo las distintas iteraciones en las que se ha dividido el proceso de desarrollo.

- **Análisis del sistema**: describe el análisis llevado a cabo al empezar cada una de las iteraciones; los requisitos recogidos, los atributos de calidad que debe tener el producto, la gestión del presupuesto, etc.
- **Diseño**: describe la etapa de diseño llevada a cabo en cada una de las iteraciones del proceso de desarrollo: el diseño de la arquitectura física y lógica, el diseño de la base de datos, y el diseño de la aplicación de Android.
- **Implementación**: desarrolla cómo ha sido implementado cada componente del sistema, tanto de los patrones de eventos, como de la aplicación en Mule ESB explicando la implementación de cada flujo de datos en la misma y del servicio REST que publica, como de la aplicación para Android.
- **Entregables** del proyecto
- **Procesos de soporte y prueba**s: habla de los procesos de soporte que se han llevado a cabo durante el desarrollo del proyecto como la toma de decisiones, la gestión de riesgos, incluyendo también las pruebas realizadas al finalizar cada iteración.
- **Conclusiones y trabajo futuro**
- **Bibliografía**

Por último, el documento incluye tres anexos:

- **Manual de instalación** tanto de la aplicación para dispositivos Android como del resto del sistema, necesario para que la aplicación funcione correctamente.
- **Manual de usuario** de la aplicación.
- **Boceto de la aplicación**

1.5. Estandarización del documento

Este documento ha sido redactado en base al estándar ISO/IEC/IEEE 16326, sobre la gestión de proyectos de ingeniería software y de sistemas. En concreto, se ha intentado adaptar la estructura de este a lo descrito en el apartado 7 del estándar, que trata de los elementos de un plan de gestión de proyecto.

Hay apartados que carecían de sentido dado el contexto del actual proyecto, desarrollado como TFG ya que el estándar está orientado a proyectos de ingeniería software y de sistemas de grandes organizaciones. Apartados como la gestión de las subcontratas, del personal, etc. se han omitido.

Además, se han tomado decisiones como incluir la bibliografía al final en lugar de al principio como indica el estándar, favoreciendo la legibilidad del documento y no distanciándonos más aún de la estructura típica de un TFG.

1.6. Acrónimos

- **CEP** Complex Event Processing
- **EPL** Event Processing Language
- **ESB** Enterprise Service Bus
- **EDA** Event-driven architecture
- **SOA** Service Oriented Architecture
- **API** Application Programming Interface

- **AMQP** Advanced Message Queuing Protocol
- **MQTT** Message Queing Telemetry Transport
- **STOMP** Simple Text Oriented Protocol
- **HTTP** Hypertext Transfer Protocol
- **REST** Representational State Transfer
- **JSON** JavaScript Object Notation
- **SQL** Structured Query Language
- **SDK** Software Development Kit
- **IDE** Integrated Development Environment
- **IoT** Internet of Things
- **SO** Sistema Operativo
- **APK** Android Application Package
- **URL** Uniform Resource Locator
- **TFG** Trabajo de Fin de Grado
- **UML** Unified Modeling Language

1.7. Definiciones

- **Procesamiento de Eventos Complejos (CEP):** Tecnología emergente que permite procesar, analizar y correlacionar grandes cantidades de eventos, para detectar y responder en tiempo real a situaciones críticas o relevantes del negocio.
- **Evento simple:** Un evento simple es una situación indivisible que ocurre en un instante concreto de tiempo.
- **Evento complejo:** Un evento complejo es una situación con mayor significado semántico que el evento simple, que nos provee de información comprensible y de gran valor, a partir de combinar y analizar otros eventos anteriormente sucedidos [3].
- **Patrón de evento:** Un patrón de evento es la definición de una serie de condiciones que al cumplirse da lugar a un evento complejo, a partir de uno o más eventos simples o de otro evento complejo.
- **Bus de Servicios Empresariales:** Es un elemento de integración de aplicaciones que permite trabajar en entornos heterogéneos, con diferentes tecnologías y protocolos. Combina servicios web, mensajería, transformación, encaminamiento y enriquecimiento de datos, etc.

2. Contexto del proyecto

En este capítulo se habla del contexto del proyecto, necesario de conocer antes de pasar a la planificación del mismo y por supuesto antes de comenzar el desarrollo. Se hará una descripción general del proyecto, se hará una relación de las características de los usuarios del sistema, se justificará el modelo de ciclo de vida elegido para el desarrollo, y se enumerarán, dando una breve explicación de en qué consisten, las tecnologías, los lenguajes y las herramientas seleccionadas para el proyecto.

2.1. Descripción general del proyecto

Se simularán una serie de datos relacionados con el tráfico, aparcamientos y polución de una ciudad mediante un simulador específico para el Internet de las cosas. Los datos simulados se enviarán a un bróker de mensajería RabbitMQ simulando que provienen de diversas fuentes. La aplicación Mule se suscribirá a los datos recibidos en el bróker mediante el protocolo AMQP 0.9.1; en dicha aplicación se programarán una serie de flujos que permitan la gestión y procesado de dichos datos para la detección de situaciones de interés a través de un motor CEP integrado en el sistema. Dichas situaciones detectadas se almacenarán en una base de datos para poder luego ser accedidas a través de una API REST. Dicha API será accedida desde la app Android desarrollada para mostrar la información que se considere relevante para los usuarios finales.

2.2. Características del usuario

Los usuarios de la aplicación se pueden dividir en dos grandes grupos:

- **Autoridades competentes:** Lo formarán los organismos públicos que necesiten la información del estado del aparcamiento, el tráfico y la polución del aire en la ciudad porque son los encargados de actuar ante las distintas situaciones perjudiciales para el medioambiente y la ciudadanía.

  Se encargarán de disolver los atascos, de aumentar excepcionalmente la bolsa de aparcamientos o de limitar la entrada de coches en cierta zona si ya no quedan aparcamientos, de aplicar ciertos protocolos si se superan los niveles de polución recomendados, etc.

  Se incluirá en este grupo al ayuntamiento de la ciudad, a la policía local, y a otras autoridades que tengan competencias en tráfico o medioambiente.

- **Resto de usuarios de la aplicación:** Lo formarán el resto de ciudadanos que decidan descargar la aplicación. Pueden tener muy diversos perfiles y usar la aplicación por distintas razones.

  Algunos, con inquietudes ecológicas, la usarán para reducir sus emisiones y estar al corriente de los niveles de contaminación de la ciudad. Otros podrán usarla para su beneficio personal, para reducir el gasto de combustible y evitar perder el tiempo buscando aparcamiento o esperando en un atasco. Y habrá otros que la usen por los ambos motivos.

Lo bueno de la aplicación es que todos los usuarios, sea cual sea su perfil y sean cual sean las razones por las que use, contribuirán a conseguir el objetivo final de la misma, que es reducir los niveles de contaminación de la ciudad y mejorar la calidad de vida de la gente.

2.3. Modelo de ciclo de vida

Elegir el modelo de cicla de vida es algo fundamental antes de emprender el desarrollo de un proyecto, ya que este definirá todos los procesos de un proyecto, desde el análisis de los requisitos hasta el final de su vida útil.

El modelo clásico en la gestión de proyectos es el modelo en cascada o modelo secuencial. En él primero, hasta que no se acaba una etapa del proceso de desarrollo no se empieza con la siguiente.

Esto tiene sus ventajas, como que la estructura del proyecto está clara, con los pasos que se deben dar de principio a fin bien definidos, es fácil de implementar y entender, y fácil de documentar.

Sin embargo, cuando hablamos de un proyecto de Ingeniería Software el modelo se muestra poco práctico, ya que los productos terminados no existen, deben estar continuamente actualizándose ya que la novedad, y la actualización y mejora continua del producto es lo que marca la diferencia entre un proyecto exitoso y un fracaso. Además, los requisitos de una proyecto de ingeniería software son cambiantes, no se pueden saber con exactitud desde un principio.

Por tanto, necesitamos un modelo fácilmente adaptable, que permita tener un producto entregable rápidamente para obtener retroalimentación y poder trabajar en mejorarlo, y así continuamente.

Por eso, el modelo de vida escogido es el modelo de vida iterativo incremental. Dividiremos el trabajo a realizar en diferentes iteraciones, y en cada iteración repetiremos todas las etapas del proceso de desarrollo: análisis de requisitos, diseño, implementación, pruebas, y documentación.

Al final de cada iteración tendremos un producto entregable, que aunque no es el producto final, aporta un nuevo valor, y puede ser evaluado para sacar nuevos requisitos para la siguiente iteración a partir de posibles mejoras, correcciones de errores, etc.

2.4. Tecnologías

En esta sección se enumerarán las tecnologías usadas durante el desarrollo del proyecto, explicando brevemente qué son, para qué sirven, y para qué han sido utilizadas en este proyecto.

2.4.1. Esper

Esper es un motor de procesamiento de eventos complejos (CEP) de código abierto basado en Java, que permite procesar y analizar grandes cantidades de eventos en tiempo real y reaccionar en base a ellos [4].

Ofrece un lenguaje de definición de patrones de eventos, Esper EPL.

2.4.2. Mule ESB

Mule ESB es un Enterprise Service Bus que permite integrar diferentes tecnologías y protocolos, permitiendo trabajar en entornos heterogéneos y combinar fuentes heterogéneas de información[5].

Integra los enfoques dirigidos por eventos (EDA) y orientado a servicios (SOA), siguiendo la arquitectura SOA 2.0, que permite que los servicios del sistema sean lanzados cuando sucede un evento.

En el proyecto desarrollado, lo utilizamos para recibir datos a través de una cola de trabajo de RabbitMQ, enviarlos en forma de eventos simples al motor CEP de Esper que generará eventos complejos que se enviarán a otra cola, se almacenarán en una base de datos, invocarán a servicios de una API, etc.

2.4.3. RabbitMQ

RabbitMQ[6] es un bróker de mensajería open source, ligero y de fácil despliegue. Aunque el principal protocolo y para el que fue originalmente implementado es AMQP, también soporta otros como MQTT, STOMP, HTTP, etc.

Lo usaremos para recibir los eventos simples simulados en una cola de entrada, desde la que recibirá los eventos nuestra aplicación. Tras procesar esos eventos, nuestra aplicación enviará los eventos complejos generados a una cola de salida también de RabbitMQ.

2.4.4. API REST

Una API REST es una interfaz que dos sistemas de computación utilizan para intercambiar información de manera segura a través de Internet[7].

La usaremos para conectar nuestra aplicación Android con el resto del sistema. Para ello, haremos uso de los métodos GET, POST, PUT, y DELETE del protocolo HTTP, enviando y recibiendo datos en formato JSON.

2.4.5. MySQL

MySQL[8] es un sistema de gestión de bases de datos relacional de código abierto, considerado el más popular del mundo.

Usaremos una base de datos MySQL en el proyecto para almacenar todos los eventos complejos generados en el motor Esper.

2.4.6. Android SDK

El SDK (Software Development Kit) de Android[9] nos provee de un conjunto de herramientas orientadas al desarrollo de aplicaciones en Android: un depurador de código, distintas bibliotecas, un emulador de dispositivos basado en QEMU, una extensa documentación, etc.

Haremos uso de estas características para desarrollar nuestra aplicación Android.

2.5. Lenguajes

En este apartado se hará una relación de los lenguajes usados, tanto lenguajes de programación como lenguajes de consultas y de procesamiento de eventos, explicando para qué son usados cada uno en nuestro sistema.

2.5.1. Java

Java [10] es el lenguaje utilizado en la mayor parte del proyecto, tanto para el servicio REST, como para la implementación de la aplicación de Android, como para desarrollar los componentes encargados de hacer funcionar el motor CEP de Esper (enviar los eventos al motor, desplegar en él esquemas y patrones, etc.).

2.5.2. SQL

SQL (Structured Query Language) es un lenguaje utilizado para administrar y recuperar información de sistemas de gestión de bases de datos relacionales como MySQL.

Ha sido utilizado para realizar las consultas de inserción y recuperación de los eventos complejos almacenados en nuestra base de datos, desde ciertos componentes de Mule ESB y desde el servicio REST.

2.5.3. Esper EPL

Es el lenguaje que ofrece Esper para declarar patrones de eventos, lo que hace es modificar y extender el lenguaje SQL para incluir operadores de patrones, ventanas de datos de eventos y temporales, etc.

Se ha utilizado para diseñar todos los esquemas y patrones de eventos usados en el proyecto.

2.6. Herramientas

En esta sección, se hablará de las herramientas utilizadas para el desarrollo de nuestro sistema, tanto los entornos de desarrollo, como el simulador usado para generar los eventos simples, como herramientas utilizadas para realizar el diseño de la aplicación, o herramientas orientadas a hacer pruebas.

2.6.1. Anypoint Studio

Anypoint Studio[11] es un IDE low-code que usando Mule ESB nos permite mediante “drag and drop” (arrastrar y soltar) desde una paleta de componentes, conectores, transformadores, etc. integrar y transformar datos de distintas fuentes, desarrollar APIs, conectarse a BBDD y muchas más funcionalidades.

Se ha usado la versión 6.6, ya que para la versión 7 no hay servidor gratuito disponible.

Es la herramienta central de nuestro proyecto, donde hemos creado los distintos flujos que se conectan entre sí y se encargan de desplegar los patrones y los esquemas en el motor Esper, recibir los eventos simples y enviarlos al motor, consumir los eventos complejos y almacenarlos en una base de datos, producir el servicio REST, etc.

2.6.2. Android Studio

Android Studio[9] es el IDE oficial para el desarrollo de aplicaciones para Android, basado en IntelliJ IDEA. Soporta Kotlin, Java y C++ como lenguajes de programación y usa Gradle como herramienta de construcción de código.

Este ha sido el entorno del desarrollo donde he desarrollado la app para Android.

2.6.3. Eclipse IDE

Eclipse es otro IDE, que aunque permite el desarrollo en muchos lenguajes de programación, se usa principalmente para el desarrollo de aplicaciones Java [12].

Se ha utilizado para desarrollar la API REST.

2.6.4. nITROGEN

nITROGEN es un sistema de generación de datos compatible con arquitecturas IoT desarrollado dentro del grupo de investigación UCASE de la UCA [13], [14].

Lo he usado para simular todos los eventos simples, que se envían a la cola de entrada del servidor RabbitMQ.

2.6.5. Balsamiq

Balsamiq es una herramienta para el prototipado de interfaces de usuario de baja fidelidad. Permite crear de manera rápida y con una curva de aprendizaje mínima bocetos de interfaces de usuario interactivas mediante “drag and drop” [15].

Ha sido utilizada para realizar el boceto de la aplicación móvil, que podemos ver en el Anexo C.

2.6.6. Postman

Postman [16] es una plataforma para implementar, usar y probar APIs. Simplifica cada etapa del desarrollo de una API REST, permitiendo generar colecciones de peticiones, implementar prueba, monitorizar el estado de una API, etc.

La hemos utilizado para realizar pruebas sobre la API implementada, como hablaremos en el apartado 8.3.3.

3. Planificación

En este capítulo se describirá la planificación seguida durante el desarrollo del proyecto.

Como ya se comentó en el apartado 2.3, para el proyecto se ha utilizado un ciclo de vida iterativo incremental.

Esto hace que el proceso de desarrollo se haya dividido en una serie de iteraciones, y en cada una de las iteraciones hayamos pasado por todas las etapas del desarrollo de un producto software: análisis, diseño, implementación y pruebas. Además, paralelamente a la ejecución de cada iteración, se ha ido documentando todo lo realizado en la presente memoria.

En el apartado 3.1 se explicará la iniciación del proyecto, cómo surgió la idea del proyecto, cómo se dio la primera aproximación a las tecnologías usadas, etc. En el apartado 3.2 se hablará de las distintas iteraciones del proyecto (excluyendo la primera aproximación).

Por último, tendremos un diagrama de Gantt en el apartado 3.3 que muestra una visión general del tiempo utilizado para cada iteración y cada etapa dentro de esa iteración.

3.1. Iniciación del proyecto

La primera aproximación a este proyecto se dio durante la asignatura de Ingeniería de Sistemas de Información. En la asignatura se aprenden la mayoría de las tecnologías utilizadas en este proyecto: el funcionamiento del motor Esper y el lenguaje Esper EPL, y el desarrollo de una aplicación mediante Mule ESB.

Como proyecto de final de la asignatura se desarrolló una aplicación usando Mule ESB, implementando una serie de patrones en Esper EPL, creando un pequeño servicio REST que cuenta los eventos que ocurren y almacenando en una base de datos todos los eventos complejos generados.

Tras reunirme con los tutores del presente TFG, decidimos que una buena opción era usar lo aprendido en la asignatura para hacer un proyecto parecido pero ampliamente extendido y pulido, dotado de mayor complejidad, y complementado con una aplicación para dispositivos Android que permitiera a los usuarios acceder a la información obtenida de los eventos complejos de una forma amigable y comprensible.

3.2. Iteraciones del proceso de desarrollo

Como ya se explicó y se justificó en el apartado 2.3, para el desarrollo del proyecto se decidió usar un modelo de ciclo de vida iterativo incremental, de modo que en cada iteración volviéramos a pasar por todas las fases del proceso de desarrollo, obteniendo al final de cada iteración un producto con un valor mayor que en la anterior.

En esta sección, se enumerarán las distintas iteraciones en las que se ha desarrollado el sistema, explicando qué se ha realizado en cada una, y hablando de cada etapa del proceso de desarrollo llevadas a cabo en cada iteración. Es decir, en cada iteración se hablará de análisis, diseño, implementación y pruebas.

3.2.1. Diseño de patrones

Una vez decidido el tema del proyecto, se procedió a pensar en los patrones a implementar. Tenían que ser patrones que nos ayudaran en la tarea de monitorizar el aparcamiento, el tráfico y la polución en la ciudades, y detectaran situaciones de interés relacionadas con estos temas.

Primero se hizo una relación de las situaciones de interés que se querían detectar, después de esta relación se diseñaron los patrones en pseudocódigo, para pasar finalmente a implementarlos usando el lenguaje Esper EPL, guardando cada patrón en un archivo .epl.

Además de los patrones para detectar los eventos complejos, se crearon por supuesto los esquemas que debían seguir los eventos simples que recibirá el motor de nuestra aplicación.

Se decidió qué parámetros debía incluir cada evento simple, tanto el de tráfico, como el de calidad de aire, como el de aparcamiento, de manera que fuera suficiente para detectar después todas las situaciones de interés requeridas a partir de ellos. Una vez hecho esto, se diseñaron en pseudocódigo y se implementaron también en EsperEPL.

Una vez implementados los patrones y esquemas, se hicieron pruebas en Esper Notebook, como desarrollaremos después en el apartado 8.3.1 de la documentación, para comprobar que eran correctos sintácticamente, y que nos ayudaban a conseguir los requisitos inicialmente identificados.

3.2.2. Implementación de los flujos en Mule

A continuación, se procedió a desarrollar lo que sería la parte central del proyecto, la aplicación desarrollada mediante Mule ESB.

Primero, se pensó en qué flujos debería tener la aplicación y qué funciones debía cumplir cada uno de ellos. Después se diseñó la estructura de todos los flujos, arrastrando desde la paleta de Anypoint Studio todos los componentes necesarios.

Una vez diseñada la estructura de los flujos, se procedió a configurar todos los componentes correctamente: configurando la conexión con la cola de entrada y la cola de salida de RabbitMQ, configurando la creación del servicio REST, configurando el envío de los eventos simples al motor de Esper y el consumo en otro flujo de los eventos complejos generados en él, configurando la conexión con la base de datos, etc.

En esta iteración, la API apenas incluía tres métodos: uno para saber si estaba en funcionamiento el servicio REST, otro para sumar 1 al número de eventos complejos cuando se generara uno y otro que devolviera el número de eventos. Además, en la base de datos se almacenaban todos los eventos, sin filtrar según fuera el tipo de estos. Todo esto se mejoró ampliamente, a la vez que se añadieron nuevos componentes para refinar todo el proceso, en la iteración descrita en el apartado 3.2.4.

Al acabar la implementación se probó todo el sistema: los eventos simples se reciben correctamente, los esquemas y patrones se despliegan correctamente, los eventos complejos se generan cuando deben generarse y se almacenan correctamente.

3.2.3. Desarrollo de la estructura de App

A continuación, se analizaron los requisitos de las funcionalidades que tendría la aplicación de Android. En base a eso, se diseñó cómo sería la navegación por la aplicación, y cómo sería la interfaz, realizando algún boceto, y eligiendo características que tendría el diseño como la paleta de colores que usaría.

Acabado todo esto, se creó un nuevo proyecto en Android Studio y se comenzaron a implementar todas las Activities que tendría la aplicación, el Navigation Drawer que se decidió que fuera un menú lateral y que permitiera navegar entre una pantalla y otra.

Sobre todo, este primer desarrollo estuvo centrado en el diseño de los layouts de las distintas pantallas de la aplicación, el desarrollo de la pantalla de bienvenida y el diseño de la navegación entre una pantalla y otra. Acabada esta iteración, nos quedó una aplicación completamente navegable, en la que se podía apreciar información de la aplicación como el logo, el icono, el objetivo, pero que no tenía ninguna funcionalidad ya que no estaba implementada la conexión con la API REST ni las tareas asíncronas que se encargarían de formatear y presentar los datos recibidos desde la API. Todo esto se mejoraría y ampliaría, completando la funcionalidad de la aplicación, en la iteración descrita en el apartado 3.2.5.

Por supuesto, la aplicación se probó, comprobando que fuera completamente navegable, y que el apartado visual fuera el esperado desde un principio.

3.2.4. Mejora del proyecto Mule y desarrollo de la API REST

En esta iteración se mejoró el proyecto desarrollado con Mule ESB, y se crearon los métodos de la API que nos servirán después para obtener los eventos complejos desde la app de Android.

Lo primero que se hizo fue analizar cómo se podía mejorar lo ya implementado, se decidió añadir un componente que comprobara si los eventos simples que se recibían en el sistema cumpliesen con el formato deseado, almacenar los eventos complejos en tablas distintas de la base de datos, según el tema de estos.

Además, se estudió qué métodos se debían implementar como parte del servicio REST para proporcionar a la aplicación en Android toda la información necesaria.

Hecho esto, se modificó la estructura de los flujos de Mule ESB, añadiendo nuevos componentes, y se procedió a implementar los nuevos métodos del servicio REST.

Al acabar la implementación, se ejecutó el proyecto Mule para ver que todo lo que se había modificado funcionaba correctamente, y mediante Postman se comprobó que los métodos de la API se comportaban como debían.

3.2.5. Mejora de la aplicación Android y conexión con la API

En esta fase del proyecto se mejoró la aplicación de Android, dotándola de funcionalidad completa y estableciendo conexión con la API REST antes implementada. De esta manera, la aplicación accede en tiempo real a los eventos complejos generados en el motor CEP, y los presenta al usuario.

Primero, se actualizará el análisis de los requisitos de la aplicación que se hizo en el apartado 3.2.3, de manera que lo que se pretende mostrar al usuario esté en sintonía con la información que recibiremos desde la API REST.

También actualizamos el diseño de las distintas pantallas para incluir la información que se recibirá desde la API, y se implementarán todas las tareas asíncronas encargadas de recuperar la información realizando una petición al servicio REST y mostrar la información de forma que sea comprensible al usuario.

Finalizada toda la implementación, realizaremos pruebas sobre el sistema completo, para comprobar que la información se muestra correctamente y que la conexión entre todos los componentes del sistema, incluida la aplicación Android, se realiza correctamente.

3.3. Diagrama de Gantt

El diagrama de Gantt (ver Figura 1) muestra toda la progresión a través del tiempo del proceso de desarrollo, mostrando tantos la duración de las distintas iteraciones del proceso, como de las diferentes etapas dentro de cada iteración.

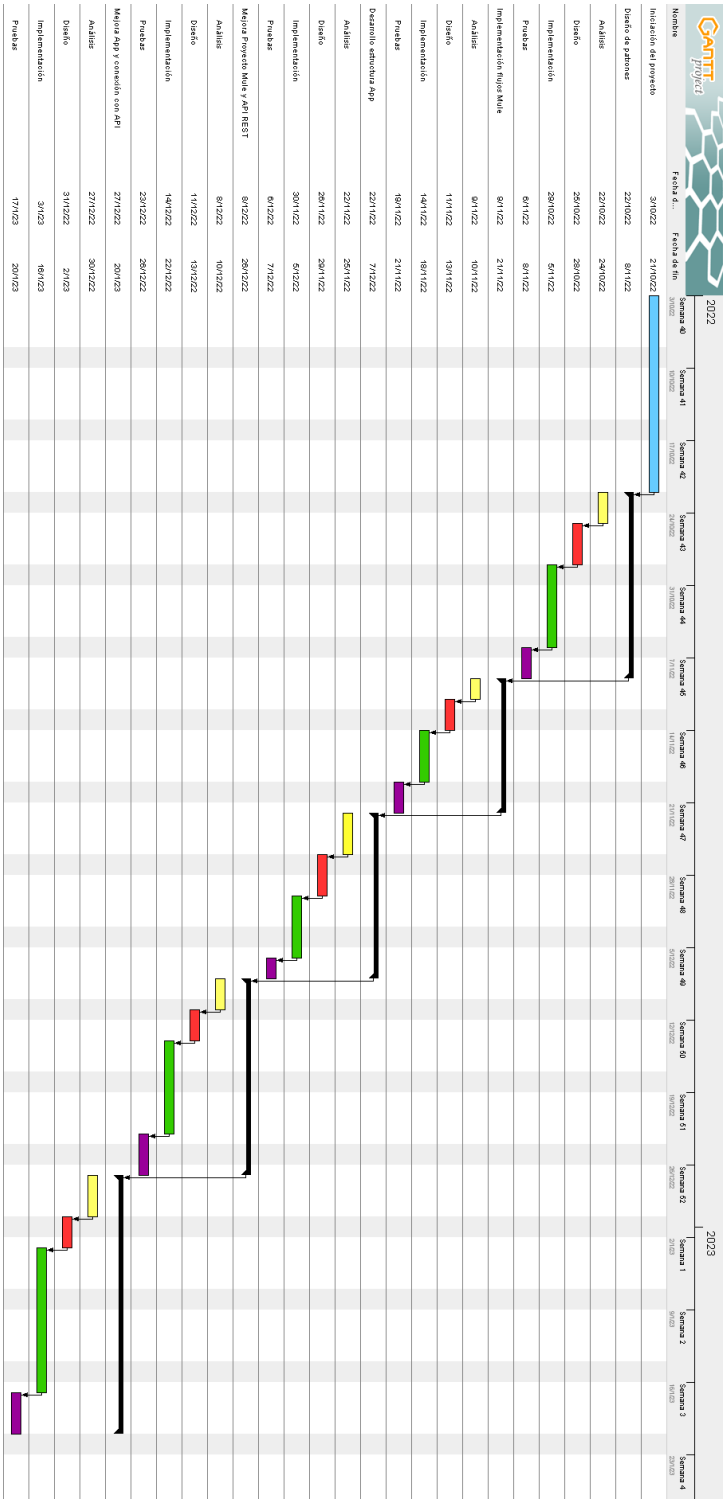

Figura 1. Diagrama de Gantt del proyecto

4. Análisis del sistema

Durante este capítulo, se realizará un análisis del sistema que vamos a desarrollar. Aunque haremos un exhaustivo análisis del sistema al empezar la primera iteración, al usar un ciclo de vida iterativo incremental, al inicio de cada iteración volveremos a analizar el sistema, extrayendo nuevos requisitos y actualizando los que sean necesarios, y actualizando el presupuesto si es necesario.

En el apartado 4.1 tendremos la especificación de requisitos, describiendo los requisitos de información y loa funcionales, modelando diagramas de casos de uso y de secuencia, haciendo una descripción de los escenarios de los casos de uso, etc.

Posteriormente, en el apartado 4.2 hablaremos de la garantía de calidad de nuestro producto, es decir, de los requisitos no funcionales que debemos tener en cuenta durante el proceso de desarrollo (seguridad, operabilidad, fiabilidad, mantenibilidad, compatibilidad…)

Por último tendremos en el apartado 4.3 la gestión del presupuesto necesario para desarrollar el proyecto, hablando de las distintas partidas y el coste de los recursos necesarios para llevar a cabo el proyecto.

4.1. Especificación de requisitos

Se realizará a continuación un análisis de los requisitos del sistema, en concreto de:

- Requisitos de información, que describen qué información deberá almacenar el sistema para lograr los objetivos marcados.
- Requisitos funcionales, que recogen las funcionalidades que debe tener el sistema si quiere cumplir sus objetivos.

Además, se describirán algunos casos de uso y se modelarán diagramas de casos de uso y de secuencia.

4.1.1. Requisitos de información

- El sistema debe almacenar en una base de datos los eventos complejos generados.
- Los eventos complejos deben ser almacenados en una tabla distinta según el evento simple con el que estén relacionados; es decir, los eventos complejos relacionados con el aparcamiento en una tabla, los relacionados con el tráfico en otra diferente, e igual con los relacionados con la polución del aire.
- El sistema debe almacenar las coordenadas de cada sensor (realmente no existen sensores ya que simulamos los eventos, pero en un entorno real sí que contaríamos con ellos).

4.1.2. Requisitos funcionales

- El sistema debe ser capaz de detectar eventos simples que recibe desde una cola de mensajes de RabbitMQ.
- El sistema debe comparar los eventos simples con una serie de patrones desplegados en el motor CEP y generar así eventos complejos, con mayor significado semántico que los primeros.

- El sistema debe enviar los eventos complejos a otra cola de mensajes de RabbitMQ y almacenarlos en una base de datos.
- El sistema debe ofrecer una API REST, que servirá para acceder a los eventos complejos generados desde la aplicación de Android.
- El sistema debe ofrecer una aplicación para Android que permita a los usuarios de la aplicación acceder a los datos sobre el tráfico, el aparcamiento y la polución de la ciudad en tiempo real.
- La aplicación de Android debe mostrar los eventos de forma amigable para el usuario, así como información detallada de cada evento.
- La aplicación de Android debe mostrar en un mapa de la ciudad los aparcamientos que se encuentran libres actualmente, permitiendo, al pulsar en el aparcamiento que se encuentra libre, ver más información sobre la plaza de aparcamiento e iniciar una ruta hacia ella mediante Google Maps.
- La aplicación de Android debe también mostrar en un mapa de Cádiz los atascos que existen en la ciudad, también mostrando más información y permitiendo establecer una ruta hacia el atasco mediante la aplicación de Google Maps.

4.1.3. Diagrama de casos de uso

En UML, un diagrama de casos de uso es una forma de representar gráficamente los distintos casos de uso del sistema, así como las relaciones entre ellos y los actores del sistema.

En nuestro caso, realizaremos un diagrama de casos de uso de la aplicación Android, como podemos ver en la Figura 2.

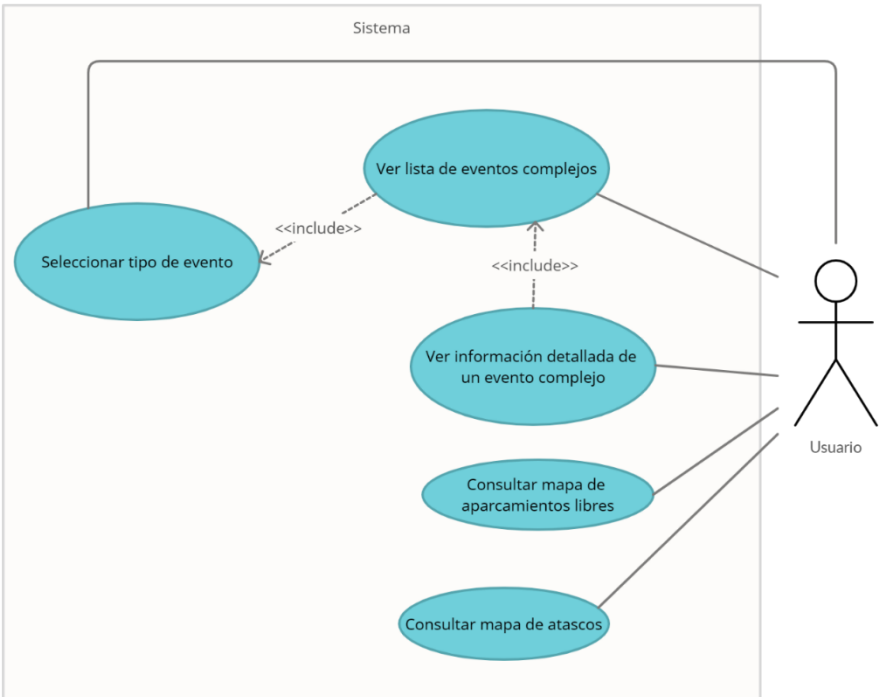

Figura 2. Diagrama de casos de uso App

4.1.4. Descripción de casos de uso

En este apartado describiremos en profundidad los casos de uso que se muestran en el diagrama, indicando los actores, las precondiciones y postcondiciones que deben de cumplirse para el correcto funcionamiento del mismo, y el escenario principal, además de ciertas extensiones de los casos de uso.

SELECCIONAR TIPO DE EVENTO

En la Tabla 1 podemos ver la descripción del caso de uso en el que un usuario elige uno de los tres tipos de eventos existentes.

Nombre	RF-01. Seleccionar tipo de evento.
Descripción	El usuario elige el tipo de evento (aparcamiento, tráfico o polución) del que desea ver la lista de eventos complejos.
Actores	Usuario.
Precondiciones	Ninguna.
Postcondiciones	El sistema se encargará de cargar la lista completa de los eventos que del tipo escogido.
Escenario principal	1. El usuario inicia la aplicación. 2. El usuario abre el menú lateral pulsando en el icono que lo abre. 3. El sistema muestra las partes del sistema a las que se puede navegar desde el menú. 4. El usuario pulsa en “Eventos”. 5. La aplicación le muestra una pantalla con tres opciones: “Aparcamiento”, “Tráfico” y “Polución”. 6. El usuario elige el tipo de evento deseado.

Tabla 1. RF-01 - Seleccionar tipo de evento

VER LISTA DE EVENTOS COMPLEJOS

La Tabla 2 muestra la descripción del caso de uso en el que se le muestra al usuario la lista de eventos complejos del tipo que haya elegido.

Nombre	RF-02. Ver lista de eventos complejos.
Descripción	El usuario elige el tipo de evento y la aplicación le muestra la lista de eventos de ese tipo que han sucedido.
Actores	Usuario.

Precondiciones	El servicio REST debe estar en funcionamiento y la aplicación debe poder enviar peticiones al mismo sin problemas.
Postcondiciones	El sistema mostrará al usuario la lista completa de eventos del tipo que eligió, indicando el nombre del evento y la fecha y hora a la que sucedió.
Escenario principal	1. El usuario inicia la aplicación. 2. El usuario abre el menú lateral pulsando en el icono que lo abre. 3. El sistema muestra las partes del sistema a las que se puede navegar desde el menú. 4. El usuario pulsa en “Eventos”. 5. La aplicación le muestra una pantalla con tres opciones: “Aparcamiento”, “Tráfico” y “Polución”. 6. El usuario elige el tipo de evento deseado. 7. El sistema muestra la lista de eventos del tipo elegido por el usuario. De cada evento le muestra el nombre, la fecha y la hora a la que tuvo lugar, y un botón para ver más información del mismo.

Tabla 2. RF-02 - Ver lista de eventos complejos

VER DETALLES DE EVENTO

En la Tabla 3 se describe el caso de uso en el que se le muestra a un usuario los detalles del evento en el que ha pulsado.

Nombre	RF-03. Ver información detallada de un evento complejo.
Descripción	La aplicación muestra al usuario información detallada de un evento.
Actores	Usuario.
Precondiciones	El servicio REST debe estar en funcionamiento y la aplicación debe poder enviar peticiones al mismo sin problemas.
Postcondiciones	Se le muestra al usuario la información detallada sobre un evento en concreto de forma amigable y comprensible.
Escenario principal	1. El usuario inicia la aplicación. 2. El usuario abre el menú lateral pulsando en el icono que lo abre.

	3. El sistema muestra las partes del sistema a las que se puede navegar desde el menú. 4. El usuario pulsa en “Eventos”. 5. La aplicación le muestra una pantalla con tres opciones: “Aparcamiento”, “Tráfico” y “Polución”. 6. El usuario elige el tipo de evento deseado. 7. El sistema muestra la lista de eventos del tipo elegido por el usuario. De cada evento le muestra el nombre, la fecha y la hora a la que tuvo lugar, y un botón para ver más información del mismo. 8. El usuario pulsa en el botón para acceder a los detalles de un evento en concreto. 9. La aplicación muestra toda la información disponible del evento.

Tabla 3. RF-03 - Ver información detallada de un evento complejo

CONSULTAR MAPA DE APARCAMIENTOS LIBRES

La Tabla 4 describe el caso de uso en el que se le presentan a un usuario en un mapa los aparcamientos que se encuentran libres.

Nombre	RF-04. Consultar mapa de aparcamientos libres.
Descripción	La aplicación muestra un mapa de Google al usuario, donde muestra los aparcamientos que se encuentran libres actualmente.
Actores	Usuario.
Precondiciones	El servicio REST debe estar en funcionamiento y la aplicación debe poder enviar peticiones al mismo sin problemas.
Postcondiciones	Se mostrará al usuario un mapa que muestre todas las plazas de aparcamiento libres en la ciudad.
Escenario principal	1. El usuario inicia la aplicación. 2. El usuario abre el menú lateral pulsando en el icono que lo abre. 3. El sistema muestra las partes del sistema a las que se puede navegar desde el menú. 4. El usuario pulsa en “Aparcamientos”.

	5. El sistema carga el mapa y se lo muestra al usuario, indicando con marcadores los aparcamientos que existen libres en la ciudad.
Extensiones	6. El usuario pulsa sobre un marcador que indica un aparcamiento libre. 6.1 La aplicación muestra un cuadro sobre el marcador con más información sobre la plaza de aparcamiento. 6.2 El usuario pulsa en el icono para ir hasta la plaza de aparcamiento libre. 6.3 Se saldrá de la aplicación y se abrirá Google Maps y se le indicará al usuario la ruta hasta el aparcamiento libre.

Tabla 4. RF-04 - Consultar mapa de aparcamientos libres

CONSULTAR MAPA DE ATASCOS

En la Tabla 5 podemos ver una descripción del caso de uso en el que se le muestran a un usuario sobre un mapa los atascos que existen actualmente en la ciudad.

Nombre	RF-05. Consultar mapa de atascos.
Descripción	La aplicación muestra un mapa de Google al usuario, donde muestra los atascos que existen en tiempo real.
Actores	Usuario.
Precondiciones	El servicio REST debe estar en funcionamiento y la aplicación debe poder enviar peticiones al mismo sin problemas.
Postcondiciones	Se mostrará al usuario un mapa que muestre todos los atascos que existen en la ciudad en tiempo real.
Escenario principal	1. El usuario inicia la aplicación. 2. El usuario abre el menú lateral pulsando en el icono que lo abre. 3. El sistema muestra las partes del sistema a las que se puede navegar desde el menú. 4. El usuario pulsa en “Atascos”. 5. El sistema carga el mapa y se lo muestra al usuario, indicando con marcadores los atascos que existen libres en la ciudad.
Extensiones	6. El usuario pulsa sobre un marcador que indica un aparcamiento libre.

	6.1 La aplicación muestra un cuadro sobre el marcador con más información sobre el atasco. 6.2 El usuario pulsa en el icono para ir hasta el atasco (esto tiene sentido por ejemplo para a policía que deba ir a devolver el tráfico a la normalidad o para una ambulancia si la causa del atasco es un accidente). 6.3 Se saldrá de la aplicación y se abrirá Google Maps y se le indicará al usuario la ruta hasta la zona del atasco.

Tabla 5. RF-05 - Consultar mapa de atascos

4.1.5. Diagramas de secuencia

En este apartado recoge distintos diagramas de secuencia que modelan la interacción entre los distintos objetos de un sistema mediante líneas de vidas o mensajes. En concreto, se han modelado tres diagramas de secuencia. El primero (ver Figura 3) recoge la interacción entre los componentes de la aplicación de Mule ESB, el segundo (ver Figura 4) la interacción necesaria para ver los detalles de un evento, y el tercero, que podemos observar en la Figura 5, la interacción necesaria entre los distintos componentes para mostrar el mapa de aparcamientos libres.

APLICACIÓN MULE ESB

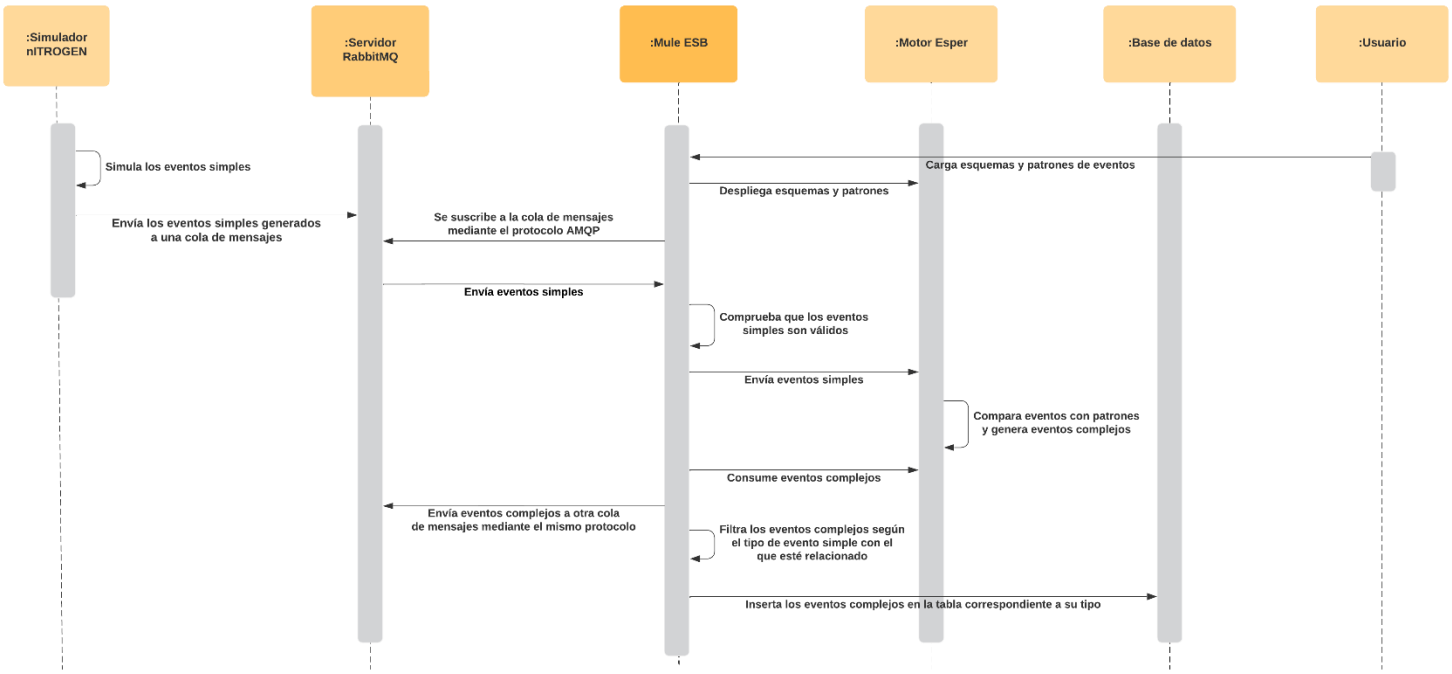

Figura 3. Diagrama de secuencia "Mule ESB"

VER DETALLES DE EVENTO

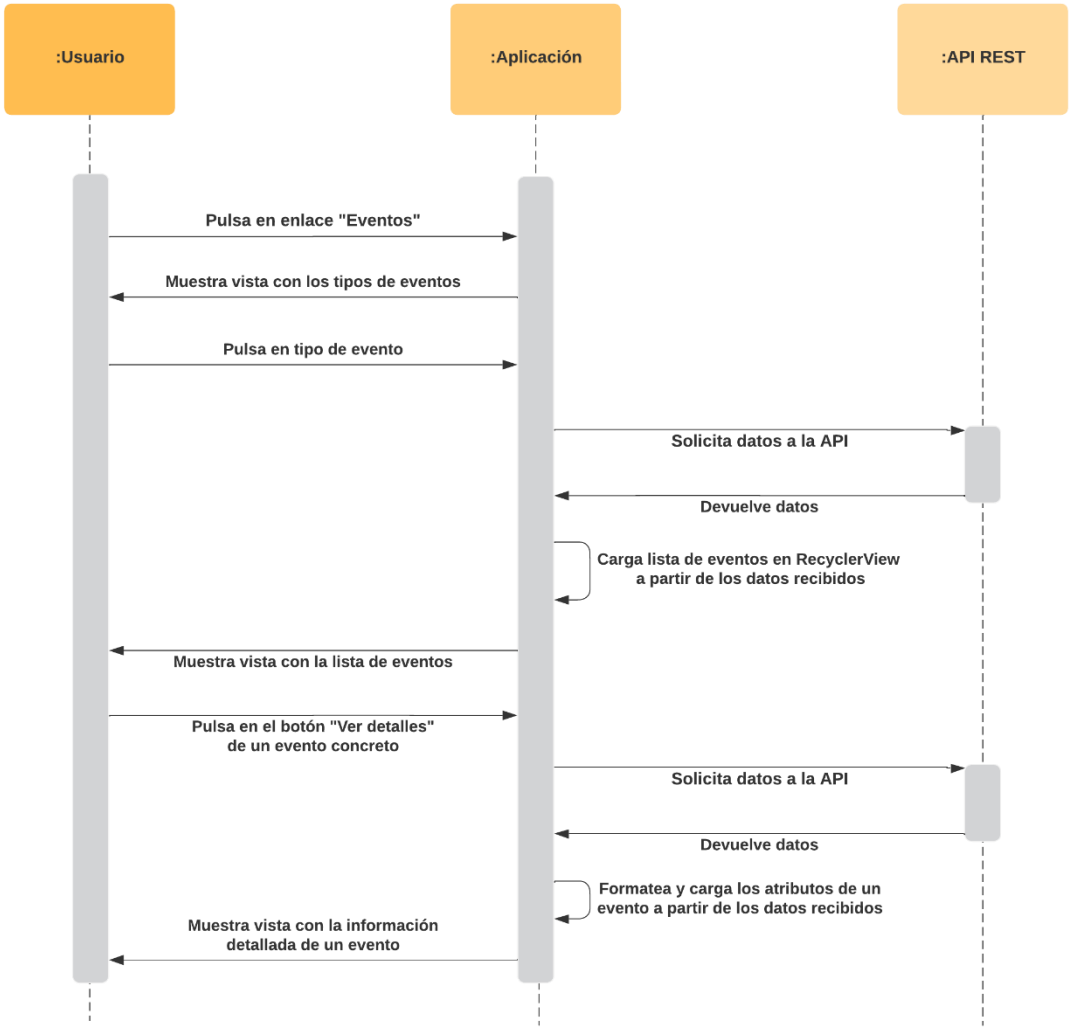

Figura 4. Diagrama de secuencia "Ver detalles de evento"

VER MAPA DE APARCAMIENTOS LIBRES

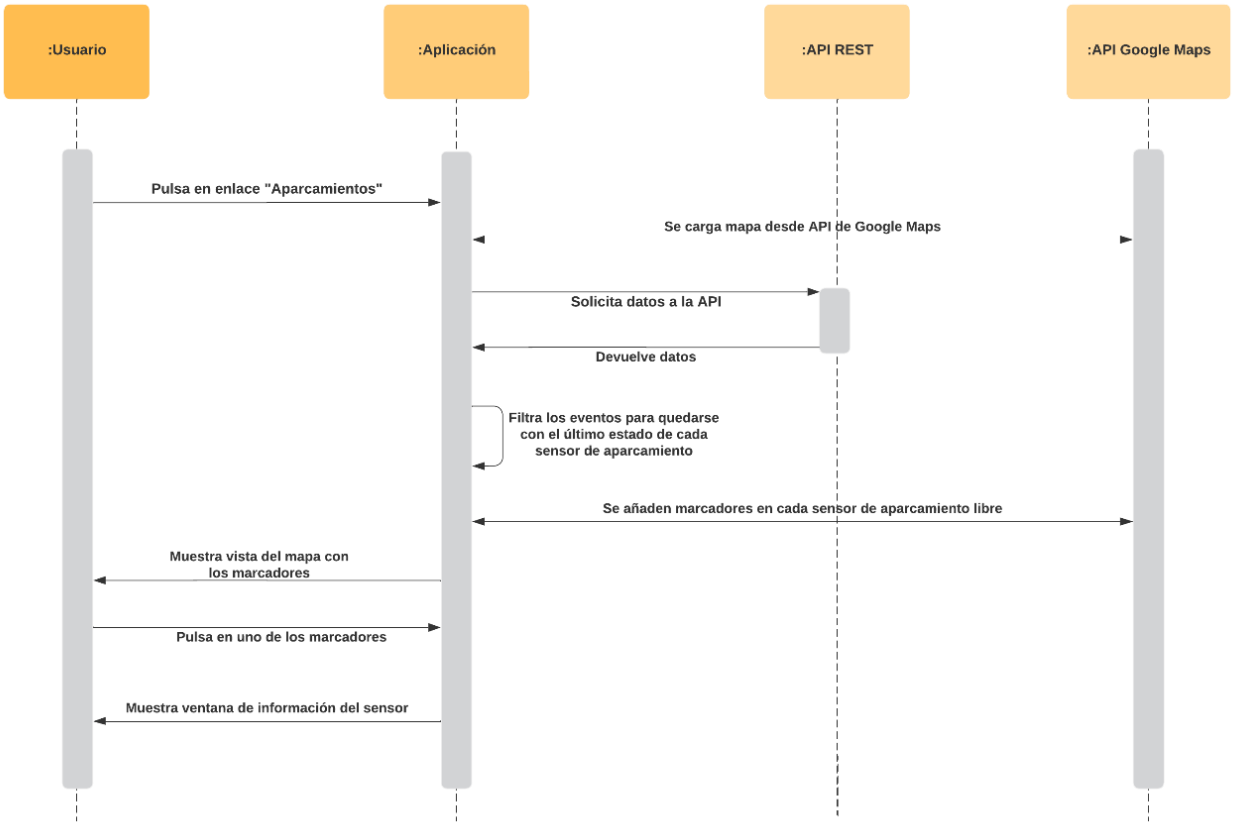

Figura 5. Diagrama de secuencia "Ver mapa de aparcamientos libres"

4.2. Garantía de calidad

En este apartado se recogerán los distintos requisitos no funcionales que garantiza la calidad del sistema. Es decir, el sistema no debe simplemente cumplir las funcionalidades anteriormente especificadas, sino que también debe tener ciertos atributos de calidad.

4.2.1. Seguridad

- El sistema deberá asegurarse de que todos los eventos que recibe siguen el formato esperado y en caso contrario descartarlos y no enviarlos al motor CEP.

4.2.2. Interoperabilidad

- El sistema deberá permitir la interoperabilidad a través de una plataforma intermedia de integración de aplicaciones, en nuestro caso Mule ESB.
- El sistema deberá ser nodo consumidor de una cola de mensajería desde la que recibirá datos, al igual que nodo productor de otra cola a la que los mandará.
- El sistema permitirá conectarse desde un dispositivo Android a través de la aplicación a un servicio REST publicado en nuestra aplicación de Mule.