UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

Identificación de desplazamientos en Ciudades Inteligentes mediante Procesamiento de Eventos Complejos

Identifying Routes in Smart Cities using Complex Event Processing

Realizado por
Carlos Salguero Tejada

Tutorizado por
Carlos Canal Velasco
Alejando Pérez Vereda

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, FEBRERO DE 2019

Fecha defensa:

Fdo. El/la Secretario/a del Tribunal

Resumen

Muchos de los estudios y proyectos dedicados a las ciudades inteligente se centran en el enfoque de una ciudad eficiente. Estas ciudades eficientes brindan sus esfuerzos en aumentar su sostenibilidad facilitándole el día a día del ciudadano. Pero para poder facilitar la vida del ciudadano es necesario orientar una arquitectura en torno a este, People as a Service (PeaaS). Esta arquitectura es un modelo de computación social y móvil que permite recopilar información a partir de los smartphones y del uso que hacen de ellos sus propietarios. Se va a hacer un estudio en el cual analizaremos la información recopilada por la señal el GPS de un smartphone. Es necesario crear un motor de incidencia capaz de transformar los datos GPS obtenidos en información de alto nivel. En este trabajo por medio de Complex Event Procesing (CEP) se crea esta herramienta capaz de analizar los datos con el fin de obtener los sucesos acontecidos durante los desplazamientos del usuario.

Palabras clave: **CEP, People as a Service, Análisis, Complex Event Processing, Desplazamiento, Smart Cities, Ciudad Inteligente.**

Abstract

Many of the studies and projects dedicated to smart cities focus on the creation of an efficient city. These efficient cities offer their efforts to increase their sustainability by facilitating the day to day of the citizen. But to facilitate the life of the citizen is necessary to guide an architecture around this, People as a Service (PeaaS). This architecture is a model of social and mobile computing that allows gathering information from smartphones and the use made of them by their owners. A study will be made in which we will analyze the information gathered by the GPS signal of a smartphone. It is necessary to create an incident engine capable of transforming the GPS data obtained into high level information. In this work through Complex Event Processing (CEP) this tool is created capable of analyzing the data in order to obtain the events that occurred during the user's movements.

Keywords: People as a Service, Analysis, Complex Event Processing, Routes, Smart Cities.

Índice

1. Introducción 1
1.1. Motivación 1
1.2. Objetivos 2
1.3. Estructura de la memoria 3

2. Tecnologías Empleadas 5
2.1. CEP 5
2.1.1. Esper & EPL 6
2.1.2. Consultas EPL 7
2.1.3. Esper-Online 9
2.1.4. Esper-Tec JAR 11
2.2. Open Street Map 13
2.2.1. Biblioteca Empleada: JMapViewer 13

3. Desarrollo del proyecto 15
3.1. Lector de CSV & Esper-Online 15
3.2. Implementación de CEP en Java 17
3.3. Mostrar análisis en mapa y correcciones 18

4. Proyecto realizado 21
4.1. Estructura del proyecto 21
4.2. CSVReader 23
4.3. EPLEngine 24
4.4. Subscriptores 25
4.4.1. BasicEventSubscriber 25
4.4.2. EndRouteSubscriber 26
4.4.3. EndRouteLastSubscriber 27
4.4.4. StartRouteSubscriber 28
4.4.5. StartRouteFirstSubscriber 28
4.4.6. RouteSubscriber 29
4.4.7. RouteTrustlySubscriber 30
4.4.8. DirectionSubscriber 31
4.4.9 ChangeDirectionSubscriber 32
4.5. Eventos 33
4.5.1. BasicEvent 33
4.5.2. InterfaceEvent 34
4.5.3. ComplexEvent 34
4.5.4. EndRouteEvent 35
4.5.5. StartRouteEvent 35
4.5.6. RouteEvent 35
4.5.7. DirectionEvent 35
4.6. Mapas 36
4.6.1. MapCreator 36

4.6.2. CoordenateList 37
5. Análisis de Resultados 39
5.1. Casos de Prueba 39
5.1.1. Caso de Prueba 1: Desplazamiento simple andando "*caminando.csv"* 40
5.1.2. Caso de Prueba 2: Desplazamiento simple en bicicleta "bicicanasta.csv" 40
5.1.3. Caso de Prueba 3: desplazamiento en coche "*Ikea.csv"* 41
5.1.4. Caso de Prueba 4: Desplazamiento en autobús y andando "*caminabus.csv*" 43
5.1.5. Caso de Prueba 5: análisis usando el metro "metro.csv" 44
5.1.6. Caso de Prueba 6: Antes y después de RouteTrustlySubscriber 46
5.1.7. Caso de Prueba 7: Desplazamiento largo en coche "MalagaCaceres.csv" 47
5.2. Fortalezas 49
5.3. Debilidades 50
6. Conclusiones y Proyectos Futuros 51
7. Referencias 53

1. Introducción

El concepto de *"Smart City"* tiene actualmente múltiples definiciones ya que es un concepto emergente con una gran cantidad de enfoques diferentes, y por eso se encuentra en continua revisión. En algunos casos incluso se habla de ciudades eficientes, las cuales se centran en adoptar un modelo basado en sostenibilidad. Hay muchas formas posibles de enfocar el desarrollo de una *"Smart City"*, pero sin duda, todos estos enfoques hablan de un bien común para los ciudadanos que habitan en ellas para facilitar su día a día.

Nosotros en este proyecto queremos orientar el concepto de *"Smart City"*, hacia el bien del ciudadano. Transformar toda la información proveniente de nuestra *"Smart City"*, en información útil para el ciudadano. Para ello, implantamos un modelo de computación
social y móvil que permite recopilar información a partir de los smartphones y del uso que hacen de ellos sus propietarios, *"People as a Service*" [1] .

Más concretamente, este proyecto se centra en el aspecto de movilidad dentro de una ciudad. Por ejemplo, imaginemos que nuestro teléfono móvil, de forma transparente y automática, detectara que podemos llegar en menos tiempo al trabajo siguiendo sus instrucciones.

En esta sección sentaremos las bases del nuestro proyecto. Hablaremos de las motivaciones de este y de los objetivos principales que nos planteamos para su elaboración. También, hacemos una pequeña descripción de los apartados que se van a tratar a lo largo de este escrito.

1.1. Motivación

En la actualidad, las ciudades inteligentes ofrecen un amplio conjunto de datos abiertos con información puntual y actualizada sobre muy diversos aspectos relevantes de las mismas. Por ejemplo, en Málaga tenemos el portal de datos abiertos del ayuntamiento (http://datosabiertos.malaga.eu/). Este portal nos provee, entre otras, información relevante sobre servicios relacionados con la movilidad y el transporte. Tales como líneas de autobús urbano, estaciones de alquiler de bicicletas públicas, niveles de ocupación de los parkings, avisos de atascos de tráfico, etc.

Pero con el enfoque actual, esta información no se traduce en un beneficio directo para sus usuarios finales que son los ciudadanos. Estos precisan que esta información sea

accesible, de fácil uso, y personalizada según sus necesidades en cada momento. Sin embargo, no disponemos de información personal de cada ciudadano que nos permita conseguir esto y, en resumen, que nos permita construir una ciudad inteligente que se adapte a sus habitantes facilitándoles su día a día mediante los servicios que ofrece. Para ofrecer esta información de forma útil y relevante nace el modelo de computación móvil (*People as a Service*) , que se basa en utilizar los sensores y recursos de computación disponibles en los teléfonos móviles actuales con objeto de representar el contexto del usuario, generando un perfil virtual del mismo [2]. Manteniendo toda esta información únicamente dentro de su teléfono, lo hacemos dueño de su información y capaz de decidir con quien la comparte o como usarla y convirtiendo el smartphone en una interfaz inteligente y personalizada para el usuario con su entorno

Para ello, nos basaremos en un TFM anterior, "Paradigma orientado a las personas para ciudades inteligentes" [3], en el que se desarrolló la aplicación móvil "*SmartUs"*, la cual recopila los itinerarios seguidos por el usuario del teléfono móvil a lo largo de sus actividades diarias como secuencias de coordenadas GPS. Pero esta información recopilada por sí misma no nos proporciona ningún dato relevante del usuario. Por lo tanto, en este TFG desarrollamos un sistema capaz de interpretar los datos recopilados por "*SmartUs"* en información útil de alto nivel.

Para hacer este análisis usamos CEP [4] una tecnología que se amolda perfectamente a nuestras necesidades, ya que es capaz de analizar conjuntamente grandes bloques de información de orígenes distintos para detectar situaciones determinadas, es decir, información de alto nivel.

1.2. Objetivos

El principal objetivo de este trabajo consiste en crear un proyecto capaz de detectar diferentes situaciones dentro de los itinerarios generados por la aplicación "*SmartUs"*. Esta aplicación móvil produce un archivo.csv que almacena trazas de localizaciones GPS seguidas por el usuario, recogidas a lo largo de los desplazamientos que haya realizado con la aplicación instalada en su teléfono. De esta forma conseguiremos que nuestro proyecto recopile información de alto nivel de los itinerarios del usuario, por ejemplo, donde empieza y finaliza un desplazamiento, cambios en la dirección o la velocidad promedio.

Mediante técnicas de Procesamiento de Eventos Complejos (CEP) analizamos los itinerarios del usuario. Este tipo de técnicas se utilizan para determinar situaciones complejas dentro de conjuntos de datos que suceden en un determinado momento, este tipo de datos son llamados eventos en CEP.

Aunque este tipo de análisis es característico de aplicaciones de procesamiento en tiempo real, al no existir actualmente un motor CEP disponible para teléfonos móviles, este análisis no puede realizar al mismo tiempo que los datos son recopilados. En su lugar, a modo de prueba de concepto, hacemos los análisis de forma posterior cuando ya disponemos de las trazas diarias recogidas en un archivo. Este análisis se realiza en un ordenador personal, con los archivos de trazas recogidas a lo largo del día. No

obstante, este estudio tiene en cuenta que el objetivo final es analizar dicha información en vivo, siendo esta una de las principales ventajas que nos proporciona CEP y una característica clave en este tipo de aplicaciones.

A continuación, narramos los principales objetivos en base al desarrollo del proyecto y los escenarios que queremos detectar en nuestro análisis.

- Implementar CEP dentro de nuestro sistema

Conocer cómo funcionan las consultas de CEP y la manera de integrar este tipo de análisis en un proyecto Java. Así como conocer las distintas bibliotecas que lo componen y la forma en la que deben relacionarse todas nuestras clases dentro de un mismo sistema.

- Determinar una parada en un lugar

Dentro de los principales objetivos en el análisis es saber en qué lugares estuvo el usuario de nuestra aplicación. Este es un punto importante para la próxima etapa del proyecto, ya que nos interesa hacer recomendaciones en base a los lugares que estuvo el usuario.

- Determinar información de un desplazamiento

Con la determinación de la inicio y fin de un desplazamiento podemos hacer más análisis dentro de este para determinar cómo fue tal desplazamiento.

- Analizar la velocidad de desplazamiento

La velocidad es una importante variable en cada desplazamiento para determinar qué medio de transporte usó el usuario.

- Una forma legible de representar nuestro análisis.

Este es uno de los puntos más clave para el análisis. Cuanto más fácil y más claros sean la forma de representar estos análisis, más fáciles serán de corregir los errores que estos tengan y así lograr un mejor resultado para nuestro proyecto

1.3. Estructura de la memoria

La estructura de esta memoria se compone por las siguientes partes:

1. Introducción:

En esta sección explicaremos el contexto del que nace este proyecto y los objetivos a alcanzar en su desarrollo.

2. Tecnologías empleadas:

Para el desarrollo de este proyecto hemos utilizado una serie de tecnologías que son explicadas en esta sección. Las principales tecnologías que abarcamos son CEP y Open StreetMap [5].

3. Desarrollo del proyecto:

Esta parte de la memoria trata las diferentes etapas del proyecto para mostrar su evolución y como se han ido cumpliendo los objetivos que hemos marcado en la Sección 1 dentro del tiempo establecido.

4. Proyecto Realizado:

Este es un apartado más técnico en el que trataremos la estructura de clases que compone el proyecto. Explicaremos el funcionamiento de cada clase y sus relaciones con las demás.

5. Análisis de resultados:

En este apartado sometemos nuestro proyecto a diferentes casos de prueba para mostrar los resultados que tiene este ante diferentes escenarios. Además, hacemos un análisis de fortalezas y debilidades de los diferentes escenarios de prueba.

6. Conclusiones y proyectos futuros:

Partiendo del análisis que hemos hecho en el apartado anterior vemos cuales son las líneas futuras de este proyecto.

2. Tecnologías Empleadas

En esta sección nos centraremos en explicar todas las tecnologías empleadas para el desarrollo de nuestro proyecto. Dando una explicación de sus funcionalidades más básicas y de las bibliotecas que hemos usado para su uso.

2.1. CEP

La principal tecnología que vamos a emplear para el desarrollo de este proyecto es CEP, que son las siglas de "*Complex Event Processing*" (Pocesamiento de eventos complejos). Se trata de un método para analizar grandes trazas de información (*Processing*), la cual, viene agrupada por sucesos denominados eventos (*Event*) con la finalidad de obtener conclusiones de estos. Además, estos análisis se pueden combinar entre sí para crear una estructura piramidal de consultas (*Complex*). Por lo tanto, con esta tecnología podemos generar un análisis complejo de un escenario mediante un conjunto de consultas que se van retroalimentando entre ellas. En definitiva, CEP es definido como una tecnología para analizar en tiempo real sucesos, teniendo en consideración el tiempo en el que sucede cada uno, para detectar situaciones de interés u obtener información de más alto nivel de complejidad o abstracción.

Un evento para CEP es un registro inmutable con una ocurrencia en el pasado de alguna acción o cambio. Las propiedades de un evento almacenan la información útil que vamos a procesar junto con un tiempo al que está anclado. Un evento puede formar parte de otros eventos, o incluso, ser parte de la condición de una regla. Esto hace que los eventos estén anidados entre sí, para facilitar el análisis de un escenario complejo.

El ejemplo más común para entender este tipo de análisis es el caso de la central nuclear. Supongamos que somos operarios de una central nuclear y queremos anticiparnos a una situación de peligro originada por un gran incremento de la temperatura debido a los procesos de fisión de átomos. La forma de anticiparnos es observando si se van dando ciertas situaciones (eventos) que puedan dar lugar a una situación de peligro. Para anticipar este estado de peligro disponemos de los tres siguientes eventos:

- Evento Monitor: Devuelve la temperatura media cada 10 segundos de un termómetro digital que mide la temperatura de la sala de fisión.
- Evento Alerta: Cada vez que el evento monitor devuelve una salida superior a 300ºC se genera un evento Alerta.

- Evento Peligro: Si se da el caso de que suceden 3 eventos Alerta consecutivos con temperaturas cada vez más altas, se crea un evento Peligro.

En este sencillo ejemplo se aprecia cómo puede conseguirse el objetivo de determinar un estado de peligro por medio del evento complejo Peligro. Hemos generado con facilidad esta regla gracias a la definición de los sucesos anteriores, los cuales, cada vez tienen una capa más de complejidad. Así, con una simple entrada de temperaturas por medio del termómetro, hemos podido detectar la situación a la que queríamos prestar atención dentro del sistema.

CEP no solo es bueno para detectar posibles estados de peligro, como hemos demostrado en el ejemplo, sino que también presenta muchas utilidades en diferentes sectores, por ejemplo, para analizar los datos de bolsa y predecir la subida y bajada de acciones, detectar rápidamente en tu comercio las compras fraudulentas on-line, y como vamos a hacer en nuestro estudio, para analizar en tiempo real los datos percibidos por sensores para generar nueva información de más alto nivel que nos permita conocer mejor la situación e interacciones de las personas para hacerles recomendaciones y ayudarlas en su día a día.

Para nuestro estudio, esta tecnología muestra características útiles como la capacidad de procesar muchos datos a gran velocidad para obtener resultados instantáneos. De esta forma, se convierte en una herramienta capaz de analizar en tiempo real nuestra localización GPS para dar consejos útiles en el momento. Otra particularidad muy importante que nos presenta esta tecnología es su baja frecuencia de computación. Gracias a ello, cumplimos con las limitaciones de los teléfonos móviles, nuestros dispositivos objetivo, para no sobrecargarlos ni agotar su batería, lo ideal es que la computación sea mínima. La posibilidad de crear nuestro analizador por medio de simples consultas nos ayuda a generar un código mucho más entendible y fácil de depurar. Y como ya hemos mencionado anteriormente, la capacidad de poder tratar una gran cantidad de datos de manera fácil y ordenada.

2.1.1. Esper & EPL

Esper [6] es un compilador en tiempo real para el procesamiento de eventos complejos y un analizador de trazas disponible para .NET y Java, además, posee una versión online dentro de la página web. Esta herramienta nos permite un rápido desarrollo para aplicaciones que procesan una gran cantidad de mensajes y sucesos ya sean recibidos en tiempo real o desde un historial almacenado. De esta información podemos determinar los sucesos de nuestro objetivo de estudio y extraer las propiedades que nos parezcan interesantes del mismo.

Además, nos proporciona un lenguaje de programación que se llama "*Event Processing Language*", de aquí en adelante abreviaremos este nombre por sus siglas EPL. Este lenguaje sigue el estándar SQL-92 e implementa expresiones para el trato de eventos a lo largo del tiempo. EPL esta compilado y ejecutado en un archivo de paquete JAR para su distribución y ejecución.

Entre las principales características que nos proporciona el motor de ejecución se encuentran una alta escalabilidad, tratamiento y gestión eficiente de la memoria, baja latencia para poder hacer grandes análisis en tiempo real sobre una gran diversidad de datos y, además, la posibilidad de analizar eventos obtenidos de un historial. El compilador y el motor de ejecución no solo pueden ser ejecutados usando un solo procesador, sino que soporta multiprocesamiento sobre varios núcleos. El propio compilador no tiene ninguna dependencia, lo cual, hace que pueda ser ejecutado bajo cualquier infraestructura o arquitectura y no necesita de ningún alojamiento externo. EPL funciona correctamente con eventos temporales y "marcas de agua" basadas en gestión del tiempo.

En cuanto a su arquitectura, está diseñada con escalabilidad horizontal. Este tipo de arquitectura proporciona al sistema flexibilidad escalar, balance y reequilibrio de carga, tolerancia a fallos, descubrimiento dinámico de nodos por medio de nodos semilla, y soporte de multicentro de datos. Toda esta arquitectura está basada en Apache Kafka [7] y Apache Zookeeper [8].

2.1.2. Consultas EPL

EPL es el lenguaje de programación que utilizamos para el análisis y creación de eventos. La sintaxis de este lenguaje sigue el estándar de consultas SQL-92, esto permite tratar a los eventos como tablas de una base de datos. La creación de eventos en EPL es lo equivalente a crear una vista en una base de datos, es decir, se crea una nueva tabla ficticia (evento) con la consulta definida en la sentencia.

Con respecto a las consultas, siguiendo el estándar SQL-92, podemos usar en nuestras sentencias los operadores clásicos INSER INTO, SELECT, FROM, WHERE, GROUP BY, LIMIT y DISTINC. También, nos permite el uso de operadores de unión como INNER-JOIN y OUTER JOIN y operadores de subconsultas como EXIST e IN.
A continuación, vamos a mostrar en la Figura 1 cómo es la estructura de estas consultas para la creación de eventos:

```
@Name('Nuevo Evento') 9
INSERT INTO [nuevo evento] 8
SELECT [propiedades del evento] 7
FROM [EVENTO_REGISTRADO] 1 || FROM PATTERN [stream] 2
WHERE [condiciones] 3
GROUP BY [agrupación] 4
HAVING [condición agrupación] 5
ORDER BY [ordenado por propiedad] 6
```

Figura 1:Consulta EPL

Ahora vamos a describir brevemente las partes de una consulta, siguiendo su orden de ejecución, el cual se ve enumerado en la Figura 1:

(1) FROM	Se define el nombre de los tipos de eventos que van a ser estudiados (en lugar de una tabla)
(2) FROM PATTERN	Puede usarse en lugar del operador FROM para definir los operadores de patrón que explicaremos más adelante, también devuelve una lista de eventos
(3) WHERE	Definimos las condiciones que deben cumplir los eventos, al igual que con las consultas SQL
(4) GROUP BY	Expresión de agrupación de eventos
(5) HAVING	condiciones para las agrupaciones
(6) ORDER BY	Expresión para ordenar la salida eventos producida
(7) SELECT	Operador donde se definen las propiedades que va a tener el evento de salida
(8) INSERT	Nombre del evento que genera la consulta
(9)@NAME	Posible anotación de la consulta

Para crear nuevos eventos complejos solo hay que generar consultas SQL de los eventos ya existentes. Cada vez que la condición de una sentencia se da, se crea un nuevo evento con las propiedades especificadas en la cláusula SELECT y el nombre de este evento será definido en la cláusula INSERT INTO.

La mayoría de las reglas propias definidas en el lenguaje EPL son tratadas en el operador FROM PATTERN. Las reglas propias de este operador son conocidas como operadores de patrón. Estas nos permiten organizar los eventos por su número de ocurrencias, orden con respecto a otros y momento en el que sucedieron. A continuación, vamos a explicar las reglas más importantes:

- **EVERY:** Se escoge cada evento especificado dentro de la cláusula
- **EVERY-DISTINCT:** Igual al operador “every”, solo que, eliminando los resultados repetidos.
- **(a) -> (b):** Esta expresión sirve para obtener el próximo evento de tipo “b” sucedido después de “a”.
- **[x:y]:** Especifica el máximo y mínimo de veces que un patrón debe suceder, donde “x” es el mínimo e “y” el máximo.
- **[N]:** Especifica el número de ocurrencias que un patrón debe tener.
- **UNTIL:** Comprueba una expresión de patrón hasta que se evalúa por verdadera.
- **WHILE:** Comprueba una expresión de patrón mientras se evalúa por verdadera.

Como hemos dicho anteriormente, EPL permite realizar consultas temporales, por lo que también proporciona reglas para su manejo. En las consultas temporales se puede indicar la unidad de tiempo en segundos, minutos, horas, días, semanas, meses y años.

- **TIMER:INTERVAL(n time):** Espera una cantidad de tiempo n antes de evaluar la condición como verdadera.
- **TIMER:AT (*, *, *, *):** Se cumple la condición como verdadero cuando es dado el tiempo especificado. Este tiempo se expresa como se muestra continuación: timer:at (minutos, horas, días del mes, mes, día de la semana [, segundos]).

- **TIMER:WITHIN (n time):** Se evalúa a falso si la expresión no se ha cumplido en un intervalo de tiempo n.

Por último, cabe mencionar la posibilidad de hacer consultas con ventanas de datos. Una ventana de datos nos permite hacer análisis de conjuntos de datos contiguos delimitados dentro de un tiempo o de un número de ocurrencias. Podemos diferenciarlas en dos tipos según el comportamiento de la ventana, deslizantes y de lote. Una ventana deslizante analiza a todos los eventos dentro del tamaño especificado empezando por un evento W1, el próximo análisis engloba el mismo tamaño de ventana, pero desde el evento W2, y así sucesivamente. En cambio, el otro tipo de análisis, los de lote, un evento solo puede ser analizado en un conjunto, de esta forma ninguna ventana puede solaparse. En la Figura 2, se puede ver claramente como las ventanas deslizantes se van solapando, mientras los de lote no.

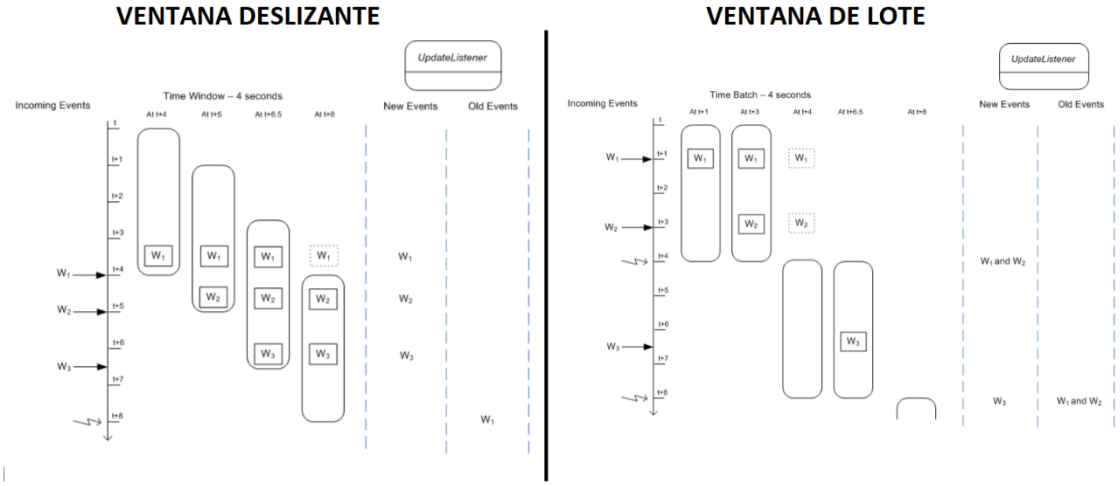

Figura 2:Ventana deslizante y ventana de lote

A continuación, se muestran los principales operadores de ventana:

- **WIN:TIME_BATCH(n time):** En esta ventana de lote retiene a todos los eventos dentro del periodo de tiempo especificado “t”.
- **WIN:LENGTH_BATCH(x):** En esta ventana de lote retiene el número “x” de eventos especificados.
- **WIN:TIME(n time):** En esta ventana deslizante se agrupa los eventos con el tamaño de tiempo especificado “t”.
- **WIN:LENGTH(x):** En esta ventana deslizante se agrupan los eventos por el número especificado “x”.

2.1.3. Esper-Online

Esper online es una herramienta web que nos permite procesar y analizar sucesos usando las consultas propias de EPL. El propósito de esta herramienta es familiarizar al usuario con el lenguaje a modo de tutorial. En esta página web no podemos usar todo el potencial que nos ofrece EPL, muchas de sus funcionalidades de escalabilidad y flexibilidad no se pueden mostrar en esta plataforma online. El principal objetivo de esta herramienta es para empezar a usar su lenguaje y comprobar como de eficaz es este tipo de análisis.

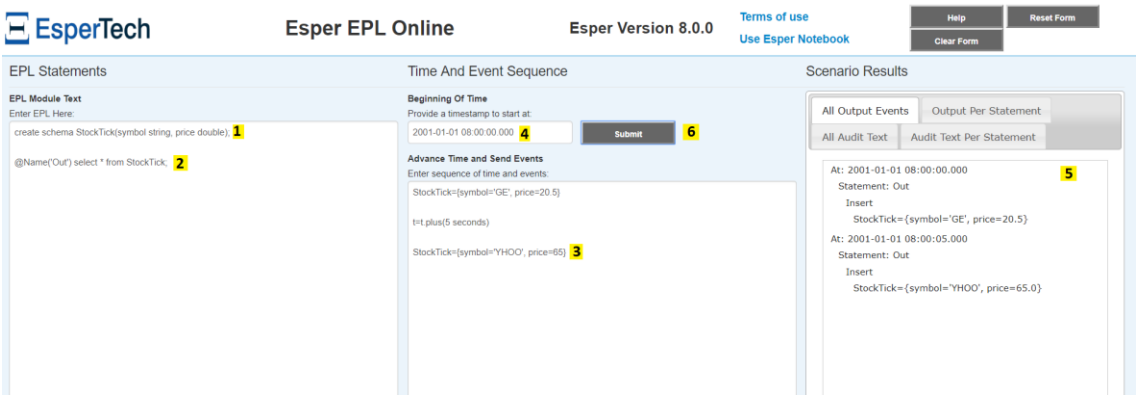

Figura 3: Esper Online

En la Figura 3 mostramos la herramienta web de Esper Online. Para utilizarla primero tenemos que definir en la ventana izquierda la estructura de los eventos (1) que serán aceptados como entrada. Justo debajo, en la misma ventana (2), escribimos todas las consultas que usaremos para procesar los datos de entrada. En la ventana central (3) se escribe en orden de ocurrencia los eventos que la herramienta web registra. También es necesario especificar el tiempo que transcurre en medio de cada ocurrencia. Justo encima de la ventana central (4) se declara la referencia temporal en el que el análisis empieza. Por último, en la ventana derecha (5), tras pulsar el botón "*Submit*" (6), se muestra el análisis que ha realiza la herramienta.

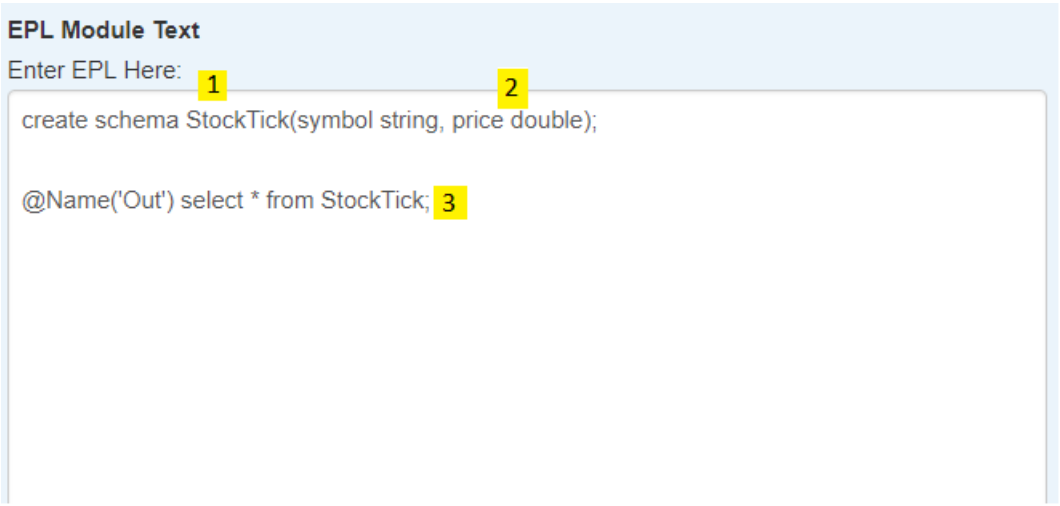

Figura 4: Ventana derecha Esper Online

La estructura básica de los eventos es descrita dentro de la Figura 4, esta es introducida en la columna de la izquierda. El nombre que lo define viene después de la cláusula CREATE SCHEMA (1). Y después del nombre entre paréntesis se declara el conjunto de propiedades que componen al evento. Cada propiedad está determinada por su nombre junto a su tipo (2). Tras esta estructura se colocan cada una de las consultas que analizan los datos que se introducen en la columna central. La estructura de las consultas (3) es la misma que se explicó en la Sección 2.1.2 de esta memoria.

Figura 5:Ventana central Esper Online

En la ventana central se definen todos los eventos que queremos analizar. Estos eventos siguen la estructura que se muestra en la Figura 5. En medio de cada uno se posiciona la diferencia temporal que los separa (1).

2.1.4. Esper-Tec JAR

La biblioteca que usaremos para implementar CEP en nuestro sistema será un archivo .jar, llamado "*EsperTech7.jar*". Esta biblioteca dispone de una guía para aprender a usarla, *"Esper Reference Documentation"* [6]. Además de la guía, también cuenta con una documentación para la API de desarrolladores donde encontramos todas las funciones y clases de las que consta este archivo, API JavaDoc [6].

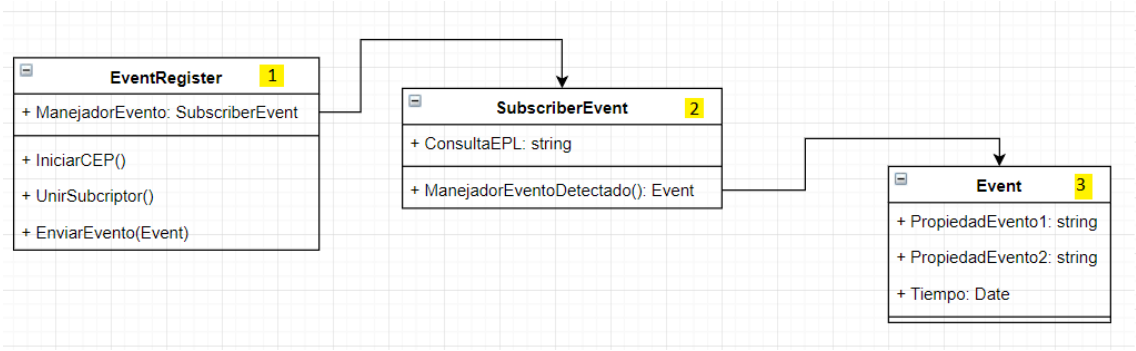

Figura 6: Estructura CEP en Java

El modelado de clases más recomendable para implementar CEP en nuestro sistema es mostrado en la Figura 6. Para empezar, tenemos que crear una clase principal que se ocupa del registro de eventos en el sistema (1). Esta clase principal está conectada con una serie de subscriptores (2), los cuales, tienen definidos las consultas EPL. Cuando las condiciones de los subscriptores se dan como válidas, su propio manejador se activa con la funcionalidad que hayamos programado. Este es un aspecto importante a la hora de implementar CEP en nuestro lenguaje de programación, ya que cuando las condiciones se dan cómo validas podemos programar la funcionalidad que queramos dentro de nuestro sistema, proporcionándole la flexibilidad comentada en apartados anteriores. Además, los eventos son definidos como clases (3). Esta propiedad permite llamar a las funciones de los eventos a la hora de generar consultas con estos. No solo podremos llamar a las funciones de los eventos sino a cualquier función de nuestro sistema, lo cual, hace mucho más fácil la creación de estas consultas.

Por último, hay que mencionar que esta biblioteca tiene dependencias y son necesarias para su correcto funcionamiento. A continuación, mostramos una tabla con las bibliotecas necesarias:

Biblioteca	Descripción	Hiperenlace
antlr4-runtime	"*ANother Tool for Language Recognition*" es un potente generador de analizadores para leer, procesar, ejecutar o traducir texto binario o texto estructurado.	https://www.antlr.org/index.html
cglib-nodep	Se utiliza para extender las clases Java e implementa interfaces en tiempo de ejecución	https://github.com/cglib/cglib/wiki
commons-compiler	La API "commons-compiler", que incluye las interfaces "*IExpressionEvaluator*", "*IScriptEvaluator*", "*IClassBodyEvaluator*" y "*ISimpleCompiler*".	https://mvnrepository.com/artifact/org.codehaus.janino/commons-compiler
janino	Janino es un compilador de Java muy pequeño y rápido ref "*Janino is a super-small, super-fast Java compiler.*"	https://janino-compiler.github.io/janino/
slf4j-api	La "*Simple Logging Facade for Java*" (SLF4J) sirve como una fachada simple o abstracción para varios marcos de trabajo de registro que permite al usuario final conectar el marco de registro deseado en el momento del despliegue.	https://www.slf4j.org/
spring	Spring se centra en el despliegue ("*plumbing*") de las aplicaciones empresariales para que los equipos puedan enfocarse en la lógica empresarial a nivel de la aplicación, sin vínculos innecesarios con entornos de implementación específicos.	https://spring.io/

2.2. Open Street Map

OpenStreetMap es un mapa editable de todo el mundo creado por voluntarios y publicado con licencia de contenido libre. Estos voluntarios para poder contribuir a la creación de este mapa han compartido la información GPS recopilada por sus dispositivos móviles. Además, para generar estos mapas se han usado capturas ortográficas de fuentes libres. Toda esta información, en conjunto con los datos vectoriales del mapa, son almacenados en una base de datos bajo una licencia libre, Licencia Abierta de Base de Datos (ODbL) [9].

OpenStreetMap posee una gran cantidad de bibliotecas, también conocidas como "*frameworks"*, que contienen código reutilizable que ayuda a los desarrolladores a integrar OpenStreetMap en sitios web o aplicaciones. Los desarrolladores proporcionan bibliotecas para acceder y analizar datos, representar mapas, geocodificar y enrutar. Estas bibliotecas son compatibles con la Web y una amplia variedad de aplicaciones de escritorio y móviles.

La finalidad de usar OpenStreetMap en este proyecto es usar las bibliotecas generadas por la comunidad para crear una representación gráfica de nuestro estudio con CEP. La representación de nuestro estudio en un mapa no solo nos facilita la depuración de nuestras consultas EPL, sino que, además, nos proporciona una herramienta ideal para representar nuestro estudio de forma visual. Una ventaja de usar OpenStreetMap es la portabilidad para poder usarlo en aplicaciones móvil, para en un futuro poder integrar esta funcionalidad dentro de la aplicación "*SmartUs*".

2.2.1. Biblioteca Empleada: JMapViewer

Esta biblioteca nos permite incorporar con facilidad la vista de un mapa OpenStreetMap dentro de una aplicación de escritorio de Java. Este componente es totalmente independiente y no necesita de otras bibliotecas.

A parte de la funcionalidad de abrir un mapa, esta biblioteca nos brinda la posibilidad de editar el mapa desplegado con diferentes puntos, iconos y polígonos. Pero si necesitáramos incorporar en el mapa otro tipo de elemento existen bibliotecas adicionales en OSM para personalizar los iconos añadidos dentro de nuestro mapa. Así nos permite añadir mayor diversidad de elementos y con más variantes de personalización. Un ejemplo muy usado por la comunidad es OpenLayer.

Por último, cabe mencionar que este mapa es editable en tiempo de ejecución, por lo tanto, cuando queramos poner en práctica nuestros análisis en tiempo real esta herramienta nos seguirá siendo totalmente útil.

3. Desarrollo del proyecto

Para la realización de este trabajo hemos usado una metodología ágil evolutiva e iterativa. Las diferentes fases del trabajo se irán realizando simultáneamente permitiendo una retroalimentación entre ellas y una corrección inmediata de errores.

En las próximas subsecciones, relatamos las principales etapas de desarrollo de nuestro proyecto, haciendo hincapié en cómo ha evolucionado sin incidir los detalles técnicos de este. Al final de cada fase descrita, a modo de resumen, se abordarán los requisitos que son cumplidos junto con los componentes desarrollados (descritos en la Sección 4) y una estimación del tiempo que se invirtió.

3.1. Lector de CSV & Esper-Online

En la primera fase de este proyecto nos centramos en desarrollar una herramienta para leer el archivo que genera la aplicación "*SmartUs*". Además de leer este fichero, creamos una función para devolver un texto de salida con el formato de Esper Online. Con este texto de salida podemos empezar a crear nuestras primeras funciones en la herramienta online.

Para empezar, creamos una clase para almacenar toda la información útil que genera la aplicación "*SmartUs*". Esta información es proveída por un archivo.csv que mostramos a continuación en la Figura 7. Esta clase, la cual llamaremos *BasicEvent*, debe almacenar los valores mostrados de latitud, longitud, velocidad y fecha.

	A	B	C	D	E	F	G	H
1	latitude	longitude	speed	date				
2	36.71314707	-4.705363621	1.335134625	Sat May 05 14:56:08 GMT+02:00 2018				
3	36.71304582	-4.705681281	1.219762325	Sat May 05 14:47:10 GMT+02:00 2018				
4	36.71282663	-4.705466862	0.622954011	Sat May 05 14:44:36 GMT+02:00 2018				
5	36.71262895	-4.705229741	1.17950201	Sat May 05 14:44:06 GMT+02:00 2018				
6	36.71241977	-4.70500243	0.920482635	Sat May 05 14:42:14 GMT+02:00 2018				
7	36.71262026	-4.705240385	1.083813906	Sat May 05 14:41:51 GMT+02:00 2018				
8	36.71284193	-4.705451352	0.569602966	Sat May 05 14:35:30 GMT+02:00 2018				
9	36.7131016	-4.705571093	1.218838573	Sat May 05 14:34:42 GMT+02:00 2018				
10	36.7132523	-4.705266655	0.756079912	Sat May 05 14:10:11 GMT+02:00 2018				
11	36.71308437	-4.705533461	0.879142463	Sat May 05 13:27:44 GMT+02:00 2018				
12	36.713219	-4.705230024	0.500291109	Sat May 05 13:21:28 GMT+02:00 2018				
13	36.71324432	-4.704889266	1.218511343	Sat May 05 13:12:08 GMT+02:00 2018				
14	36.71330922	-4.705219911	0.946056724	Sat May 05 13:09:59 GMT+02:00 2018				

Figura 7: Archivo.csv "SmartUs"

Por otro lado, creamos nuestra clase *CSVReader*, la cual, se ocupa de inyectar la información del archivo en una lista de *BasicEvent*. Además, dentro de esta clase implementamos un método para mostrar la lista en el formato requerido por Esper Online, así, podemos empezar a hacer nuestros primeros análisis, como se puede ver en la Figura 8.

```
BasicEvent ={ timestamp = "Sat May 05 12:08:38 CEST 2018", latitud = 36.685184, longitud = -4.4459815,Speed = 4.5937195 }

t = t.plus(2 seconds)
BasicEvent ={ timestamp = "Sat May 05 12:08:40 CEST 2018", latitud = 36.685387, longitud = -4.4462647,Speed = 5.4081 }

t = t.plus(2 seconds)
BasicEvent ={ timestamp = "Sat May 05 12:08:42 CEST 2018", latitud = 36.686066, longitud = -4.447264,Speed = 5.709785 }

t = t.plus(1 seconds)
BasicEvent ={ timestamp = "Sat May 05 12:08:43 CEST 2018", latitud = 36.68629, longitud = -4.447573,Speed = 4.5272927 }

t = t.plus(8 seconds)
BasicEvent ={ timestamp = "Sat May 05 12:08:51 CEST 2018", latitud = 36.687103, longitud = -4.449043,Speed = 1.9880025 }

t = t.plus(22 seconds)
BasicEvent ={ timestamp = "Sat May 05 12:09:13 CEST 2018", latitud = 36.687317, longitud = -4.4488077,Speed = 4.702019 }

t = t.plus(5 seconds)
BasicEvent ={ timestamp = "Sat May 05 12:09:18 CEST 2018", latitud = 36.687595, longitud = -4.4487014,Speed = 4.952937 }

t = t.plus(56 seconds)
BasicEvent ={ timestamp = "Sat May 05 12:10:14 CEST 2018", latitud = 36.687862, longitud = -4.448449,Speed = 8.122175 }

t = t.plus(3 seconds)
BasicEvent ={ timestamp = "Sat May 05 12:10:17 CEST 2018", latitud = 36.688156, longitud = -4.4483314,Speed = 10.167044 }
```

Figura 8: Conversion del archivo.csv para Esper Online

Con esta herramienta online, además de empezar a entender el lenguaje EPL, creamos nuestras primeras consultas para devolver todos los eventos enviados y para determinar el fin de un desplazamiento.

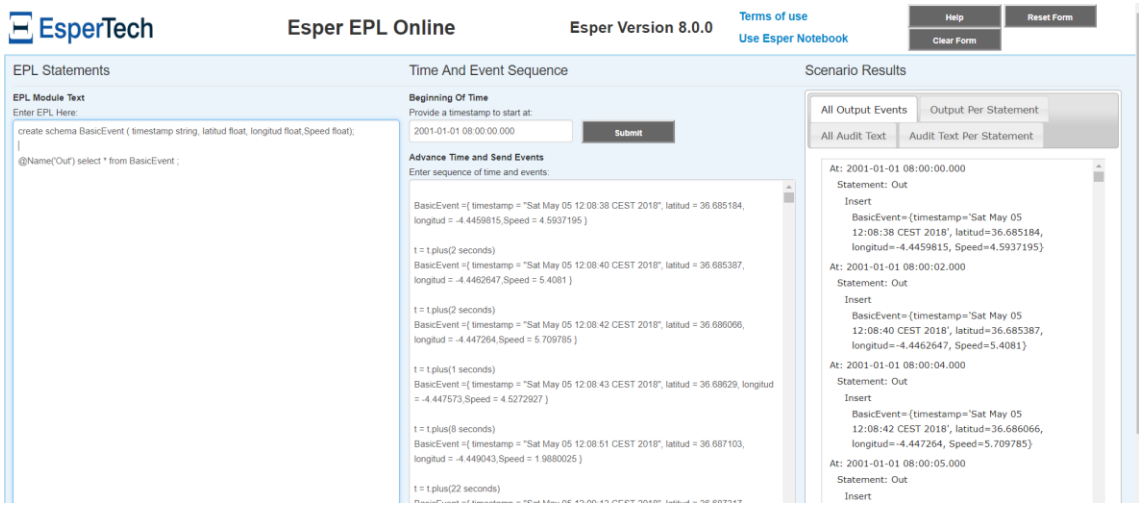

Figura 9: Esper Online con datos de “SmartUs”

En esta fase del desarrollo fueron invertidas 3 semanas. Este periodo de tiempo fue invertido para empezar conocer los fundamentos para crear consultas EPL, se desarrollaron las clases *CSVReader* de la Sección 4.2, y la clase *BasicEvent* de la Sección 4.5.1. Además, se crearon las consultas para devolver todos los *BasicEvent* y reconocer el fin de un desplazamiento para la versión Esper Online.

3.2. Implementación de CEP en Java

Entendiendo mejor la formación de consultas EPL mediante su herramienta Online, el siguiente paso es integrar CEP en nuestro programa, con la biblioteca "*esper-7.1.0.jar*". Esta mejora nos permite ser independientes a la herramienta online y les ofrece a las consultas la flexibilidad de un lenguaje de programación como Java.

En la documentación no solo encontramos tutoriales de cómo implementar Esper en nuestro código, sino que también la estructura más recomendable de hacerlo. Como hemos descrito en la Sección 2.1.4, teniendo un manejador principal el cual está unido a varios subscriptores atentos a sus diferentes consultas.

En esta fase los subscriptores son programados para devolver una cadena de texto cada vez que su consulta detecta un nuevo evento dentro del análisis. En la Figura 10 podemos ver un breve ejemplo de cómo nuestro sistema imprime por pantalla los eventos que detecta del análisis.

```
- [Cambio de direccion]:
- [Cambio de direccion]:
- Coordenadas de cambio...
- Latitud = 36.687103
- Longitud = -4.449043
- Tiempo = Sat May 05 12:08:51 CEST 2018
- Pi = 0.7853981633974483
- Tiene que ser mayor = 1.6902146

- [Cambio de direccion]:
- [Cambio de direccion]:
- Coordenadas de cambio...
- Latitud = 36.68843
- Longitud = -4.448248
- Tiempo = Sat May 05 12:10:21 CEST 2018
- Pi = 0.7853981633974483
- Tiene que ser mayor = 1.4285454

- [Cambio de direccion]:
- [Cambio de direccion]:
- Coordenadas de cambio...
- Latitud = 36.69156
- Longitud = -4.45406
- Tiempo = Sat May 05 12:12:57 CEST 2018
- Pi = 0.7853981633974483
- Tiene que ser mayor = 1.3628572
---------------------------------
- [Fin desplazamiento]:
 latitud = 36.692
 longitud = -4.453551
 speed = 2.9625952
 time = Sat May 05 12:13:13 CEST 2018
---------------------------------
---------------------------------
```

Figura 10: Texto de salida del análisis CEP

Durante esta etapa de desarrollo de 4 semanas se implementó la clase *EPLEngine* de la Sección 4.3, la cual se ocupa de manejar todos los eventos, los subscriptores *BasicEventSubscriber*, *EndRouteSubscriber*, *EndRouteLastSubscriber*, *StartRouteSubscriber*, *StartRouteFirstSubscriber* y *RouteSubscriber* (Sección 4.4), y los eventos *InterfaceEvent, ComplexEvent, EndRouterEvent* y *StartRouteEvent* (Sección 4.5). Todas estas implementaciones fueron necesarias para integrar CEP dentro del proyecto y se consiguió asentar la base para que las consultas fuesen capaces de detectar de inicio a fin un desplazamiento completo, concediéndole al sistema la capacidad de detectar en que lugares se detuvo el usuario.

3.3. Mostrar análisis en mapa y correcciones

En la anterior fase de desarrollo aprendimos que la información que muestra el programa era muy difícil de entender y depurar. Pensamos que la mejor solución para mostrar nuestros análisis es por medio de un mapa en el cual posicionar los eventos que suceden. De forma que, una vez nuestro programa analiza la información leída del archivo.csv aparece un mapa indicando la localización de los eventos sucedidos.

Para implementar la salida del mapa buscamos una opción de código abierto que nos permita desplegar un mapa donde poder mostrar la posición de los diferentes sucesos que detectamos. Explorando las bibliotecas creadas con OSM encontramos JMapViwer la cual nos devuelve un mapa totalmente editable donde podemos posicionar con diferentes colores la posición de los sucesos que detectábamos.

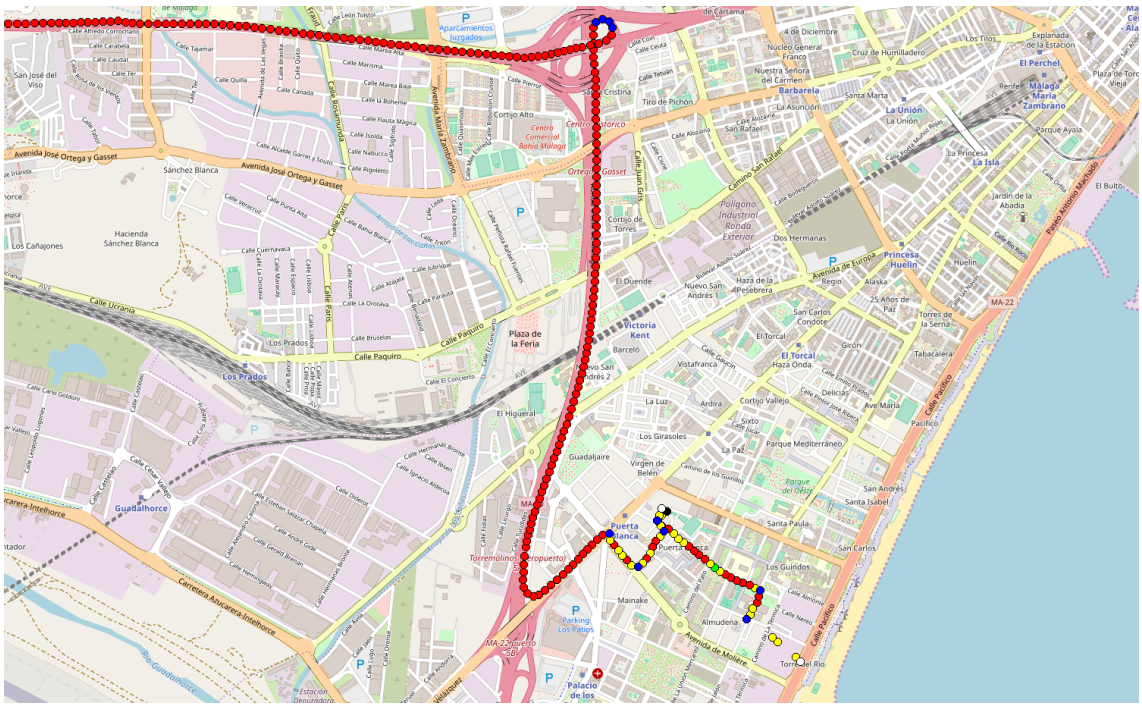

Figura 11:Mapa del análisis CEP

Nuestros resultados que antes representábamos por una salida de texto indicando los eventos complejos que íbamos descubriendo ahora son representados por un mapa. Este mapa, como podemos ver en la Figura 11, esta dibujado con una serie de puntos los cuales según su color representa un evento u otro de los que hemos programado en nuestro análisis.

Con la incorporación del mapa se facilitó la tarea de detectar y corregir errores de las sentencias EPL que ya habíamos creado y de crear nuevas sentencias de mayor complejidad. Por ejemplo, corregimos la sentencia de los desplazamientos para hacerlos más fiables y pudimos crear sentencias para detectar cambios bruscos de dirección y mostrar en el mapa la velocidad de desplazamiento en función a una gama de colores.

Finalmente, en esta etapa de 3 semanas se desarrolló un método para mostrar con claridad los resultados del análisis con la implementación de las clases *MapCreator* y

CoordenateList, de la Sección 4.6, mediante un Mapa. En este mapa se representó de forma clara la velocidad que llevaba el usuario durante la recogida de datos de "*SmartUs*" mediante colores en función de la velocidad. Una vez el mapa fue implementado se facilitó mucho la tarea de crear nuevas consultas dando lugar al desarrollo de los subscriptores *RouteTrustlySubscriber*, *DirectionSubscriber* y *ChangeDirectionSubscriber* (Sección 4.4) y los eventos *RouteEvent* y *DirectionEvent* de la Sección 4.5.

4. Proyecto realizado

A continuación, explicaremos de forma más técnica el sistema final que se ha desarrollado a lo largo de este trabajo. Explicaremos cuales son las diferentes componentes que hemos desarrollado y su funcionamiento e integración.

4.1. Estructura del proyecto

Para entender mejor cómo se relacionan todos los componentes de nuestro proyecto hemos desarrollado un diagrama de clases. Este diagrama mostrado a continuación es una versión simplificada para entender mejor la estructura del proyecto y la relación que hay entre todos sus componentes. Hemos excluido todos los métodos constructores, *get* y *set* dentro del diagrama de la Figura 12.

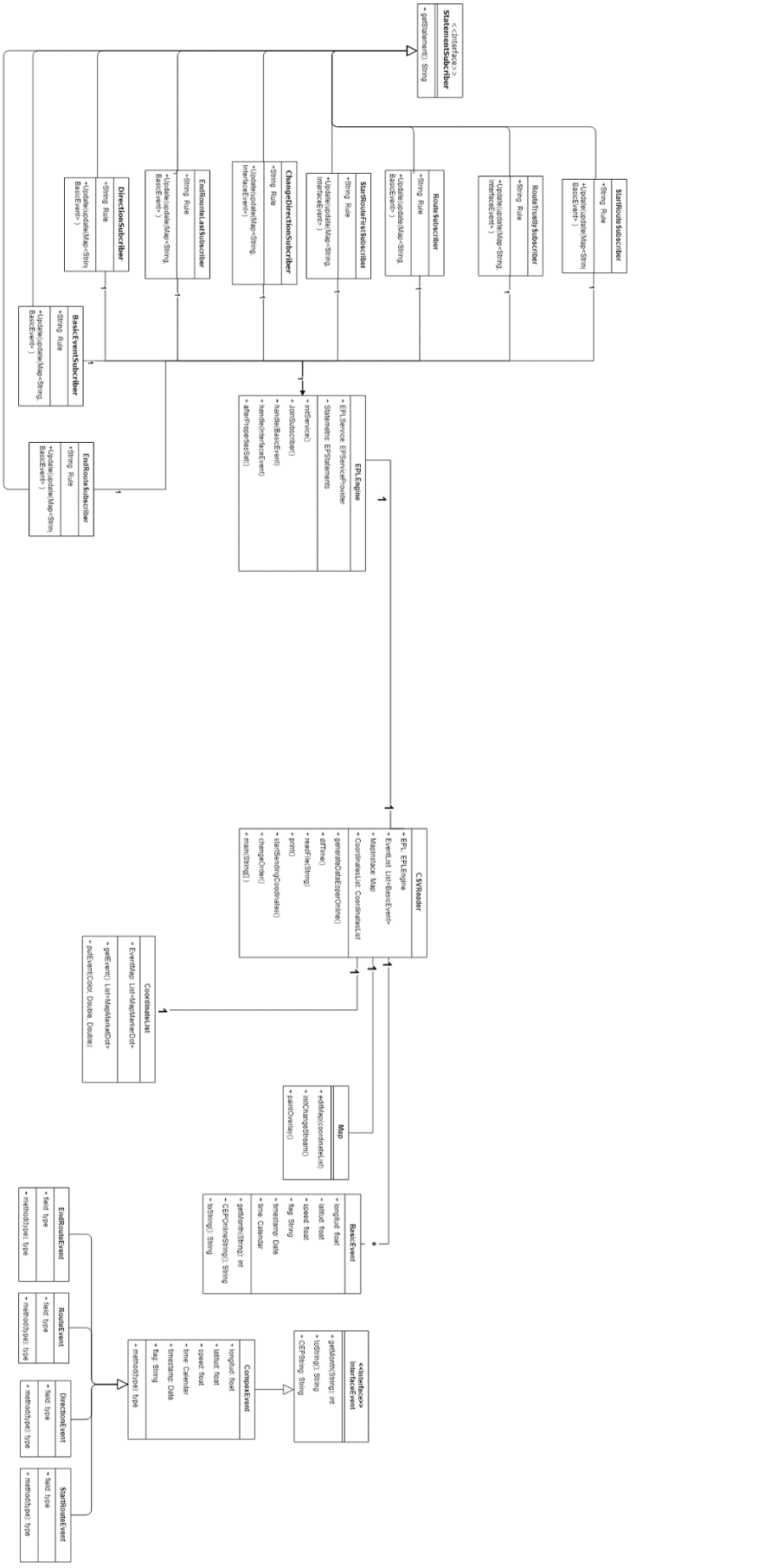

Figura 12: Diagrama de clases

Ahora vamos a describir a alto nivel como es el ciclo de vida del sistema que se ha desarrollado en este proyecto. Para ello nos ayudaremos de la Figura 13 en la que replicamos las principales funciones que se desarrollan.

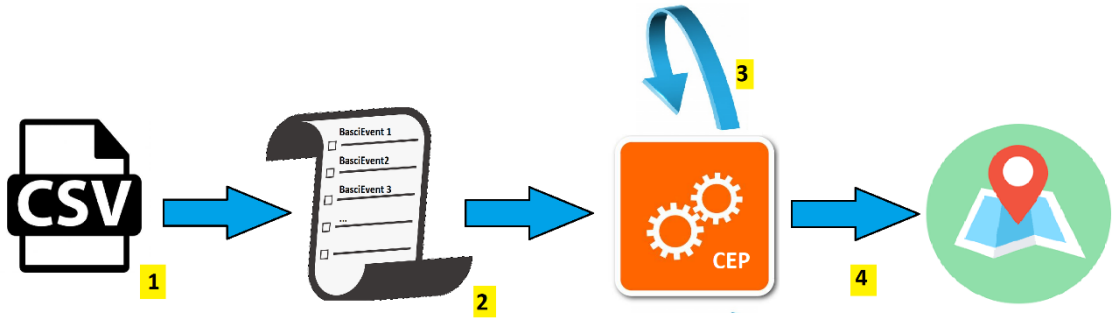

Figura 13: Ciclo del vida del proyecto

Para empezar, necesitamos recolectar toda la información del archivo.csv. Para esto, nuestra clase *CSVReader* lee el fichero y almacena cada línea como un evento de la clase *BasicEvent* (1). Una vez es leído este fichero es la hora de enviar todos estos eventos a nuestro analizador CEP, instanciado en la clase *EPLEngine* (2). Este analizador recibirá cada evento con una marca de tiempo del instante en el que sucedió. A medida que vaya recibiendo eventos, nuestro analizador irá comprobando las consultas que tiene programadas para detectar nuevos eventos (3). Cada vez que una consulta detecta un nuevo evento, guardamos las coordenadas del suceso en una lista junto a un color. Una vez acaba el análisis, desplegamos un mapa junto con todos los eventos detectados dibujados en él cómo puntos (4). Cada punto tendrá un color diferente en función del tipo de evento que se haya detectado. Finalmente, podemos apreciar en el mapa mencionado el análisis realizado por nuestro sistema.

4.2. CSVReader

La clase *CSVReader* es la que se ocupa de leer el archivo.csv generado por la aplicación "*SmartUs*". La función principal de esta clase es transformar cada línea del fichero en un evento computable para nuestro analizador CEP (clase *EPLEngige.java*).
Para realizar esta transformación hemos leído cada línea del archivo como una cadena de texto sabiendo que cada objeto diferente esta limitado con el carácter ";". Por lo tanto, hemos transformado la cadena de texto recibida en las propiedades que hay dentro de *BasicEvent*. Cabe mencionar que este programa al inicio de su ejecución, como se ve en la Figura 14, le pedirá una entrada de texto para encontrar el archivo que se va a leer. Estos archivos se encuentran en la carpeta *files* dentro del proyecto.

```
compile:
run:
Intruduzca el nombre del archivo dentro de la carpeta files
Si pulsa intro se leera la ruta predeterminada
...
```

Figura 14: Texto inicio del analizador

Dentro de esta clase, además de implementar las funciones de lectura, también encontramos las relacionadas con las utilidades para el tratamiento de eventos. Con relación a los eventos registrados del fichero tenemos dos funciones principales, una *generateDataEsperOnline*, para devolver una cadena de texto que contenga todos los eventos en un formato compatible con Esper Online, y otra *startSendingCoodinates*, para enviar cada evento registrado al analizador CEP desarrollado dentro de nuestro sistema.

En un futuro cuando la arquitectura CEP esté integrada en "*SmartUs*", no será necesario generar ningún archivo.csv de salida, ni siquiera la existencia de esta clase. Directamente la aplicación enviará los eventos sucedidos a nuestro analizador CEP.

4.3. EPLEngine

Motor principal de la lógica CEP, cuya principal funcionalidad es manejar todos los eventos que queremos analizar con nuestras consultas EPL. Para configurar correctamente nuestro analizador es necesario hacer una serie de configuraciones que explicamos en esta sección.

Para empezar, en nuestra clase *EPLEngine* establecemos la configuración por defecto y a continuación le especificamos el nombre de la carpeta que contiene nuestros eventos. Así, en nuestras consultas podemos acceder a sus propiedades como si fueran objetos, es decir, haciendo una llamada a sus funciones.

```
public void initService() {

    Configuration config = new Configuration();
    config.getEngineDefaults().getThreading().setInternalTimerEnabled(false);
    config.addEventTypeAutoName("Event");
    EPLService = EPServiceProviderManager.getDefaultProvider(config);

    JoinSubcribers();
}
```

Figura 15: Función initService

Por cada consulta EPL que queramos hacer en nuestro análisis debemos proporcionar un objeto manejador, el cual, en el caso de que la consulta se dé por verdadera deberá de realizar alguna acción. Estas clases son llamadas subscriptores, ya que están atentos a la clase principal que administra la entrada de eventos. En la Figura 16 mostramos

todos los subscriptores que tiene nuestro analizador. Todas estas clases subscriptor extienden de la misma interfaz, para poder manejarlos con las mismas funciones.

```
private StatementSubscriber BasicEventSubscriber;
private StatementSubscriber EndRouteSubscriber;
private StatementSubscriber StartRouteSubscriber;
private StatementSubscriber StartRouteFirstSubscriber;
private StatementSubscriber EndRouteLastSubscriber;
private StatementSubscriber RouteSubscriber;
private StatementSubscriber RouteTrustlySubscriber;
private StatementSubscriber DirectionSubcriber;
private StatementSubscriber ChangeDirectionSubscriber;
```

Figura 16: Subscriptoes del sistema

Por último, debemos crear una función para introducir eventos a nuestro analizador. En nuestro caso, hemos creado dos y ambas son llamadas *handle*. Uno para los eventos básicos de nuestro análisis, *BasicEvent*, y otro para los demás eventos complejos que obtenemos a partir de las consultas. Es necesario y muy importante proporcionar el instante del suceso antes de enviar el evento, ya que de lo contrario todos sucederán en el mismo instante y perderíamos la variable del tiempo en el análisis. En la Figura 17 mostramos como se configura la entrada de eventos.

```
public void handle(BasicEvent event) {
    //CurrentTimeEvent
    EPLService.getEPRuntime().sendEvent(new CurrentTimeEvent(event.getDateTime().getTime()));
    EPLService.getEPRuntime().sendEvent(event);

}
public void handle(InterfaceEvent event) {
    EPLService.getEPRuntime().sendEvent(new CurrentTimeEvent(event.getDateTime().getTime()));
    EPLService.getEPRuntime().sendEvent(event);
}
```

Figura 17: Función handle

4.4. Subscriptores

Como se ha explicado en la sección anterior los subscriptores son la lógica del análisis. Se ocupan tanto de las consultas, como de la funcionalidad que tengan cuando se den sus condiciones como ciertas. Esta funcionalidad se implementa en su función *update*. Cada subscriptor debe estar conectado a nuestra clase *EPLEngine*.

A continuación, explicaremos las consultas que hemos establecido en cada subscriptor junto con la funcionalidad que desempeña su función *update*. Cada descripción ira acompañada de una imagen del componente en sí, la cual carecerá de título.

Cabe mencionar que todos los subscriptores extienden de una clase abstracta para que estos puedan ser tratados por las mismas funciones. Esta clase abstracta la hemos nombrado *StatementSubscriber*.

4.4.1. BasicEventSubscriber

Subscriptor para devolver todos los eventos del tipo *BasicEvent*.

- **Consulta**

```
private final String  Query =
        "select a "
      + "from BasicEvent as a";
```

Figura 18

La consulta de este subscriptor es la más simple de todas, simplemente devuelve todos los eventos del tipo *BasicEvent*.

- **Update**

```
public void update(Map<String, BasicEvent> eventMap) {
    BasicEvent event = (BasicEvent) eventMap.get("a");

    StringBuilder sb = new StringBuilder();
    sb.append("---------------------------------");
    sb.append("\n- [PuntoEvent]: ");
    sb.append("\n latitud = " + event.getLatitud() );
    sb.append("\n longitud = " + event.getLongitud() );
    sb.append("\n speed = " + event.getSpeed() );
    sb.append("\n timestamp = " + event.getTimestamp().toString() );
    sb.append("\n---------------------------------");
    System.out.println(sb);

    if(event.getSpeed() < SlowSpeed){
        CSVReader.CoordinatesList.putEventMap(Color.yellow, event.getLatitud(), event.getLongitud());
    }else if(event.getSpeed() >= SlowSpeed && event.getSpeed() < FastSpeed){
        CSVReader.CoordinatesList.putEventMap(Color.orange, event.getLatitud(), event.getLongitud());
    }else if(event.getSpeed() >= FastSpeed && event.getSpeed() < FakeSpeed){
        CSVReader.CoordinatesList.putEventMap(Color.red, event.getLatitud(), event.getLongitud());
    }

}
```

Figura 19

El evento registrado por la consulta es devuelto por la pantalla de la consola con sus propiedades más importantes junto con su nombre. Además, almacenamos dentro de la variable *CoordinatesList* un punto latitud-longitud junto a un color en función de su velocidad. Esto, a la hora de dibujarlo en nuestro mapa, nos ayuda a tener una idea de la velocidad que llevaba. Para velocidades lentas usamos el amarillo, para las medias naranja y para las rápidas el rojo.

4.4.2. EndRouteSubscriber

Consulta para detectar el final de un desplazamiento. Como mencionamos anteriormente, "*SmartUs*", solo introduce una nueva fila en el archivo.csv si el usuario se ha desplazado una distancia de 35 metros. Por lo tanto, la situación que vamos a obtener en la consulta es cuando se deja de enviar coordenadas GPS durante un tiempo porque el usuario se encuentra detenido en un espacio.

- **Consulta**

```
private final String  Query = "select a1 "
        + "from pattern [every a1 =   BasicEvent -> "
        + "(timer:interval(150 seconds) and not a2 = BasicEvent)]";
```

Figura 20

Esta consulta se activa cuando después de un *BasicEvent* no detectamos el siguiente (*BasicEvent*) hasta que ha pasado un intervalo de tiempo superior a 150 segundos.

- **Update**

```
public void update(Map<String, BasicEvent> eventMap) {

        BasicEvent event = (BasicEvent) eventMap.get("al");

        StringBuilder sb = new StringBuilder();
        sb.append("---------------------------------");
        sb.append("\n- [Fin desplazamiento]: ");
        sb.append("\n latitud = " + event.getLatitud() );
        sb.append("\n longitud = " + event.getLongitud() );
        sb.append("\n speed = " + event.getSpeed() );
        sb.append("\n time = " + event.getTimestamp().toString());
        sb.append("\n---------------------------------");
        System.out.println(sb);
        CSVReader.EPL.handle(new EndRouteEvent(event));
}
```

Figura 21

La operación de activación de esta consulta imprime por la consola del terminal las principales propiedades de una detención. Además, se envía al objeto *EPLEngine* el evento, del tipo *EndRouteEvent*, detectado para que se tenga en cuenta en el análisis.

4.4.3. EndRouteLastSubscriber

Analizando nuestra anterior consulta nos dimos cuenta de que no detectábamos un caso concreto de detención. Este caso era el último evento almacenado en nuestro archivo.csv. Para tener en cuenta este evento como una detención le pusimos la etiqueta “fin” en la propiedad *flag*.

- **Consulta**

```
private final String  Query = "select al " +
            " from BasicEvent as al where flag = \"fin\" ";
```

Figura 22

Básicamente, devuelve todos los *BasicEvent* que contengan en la propiedad *flag* la cadena de texto “fin”.

- **Update**

Como lo detectado en esta operación se trata de un *EndRouteEvent*, su operación *update* es la misma que la del subscriptor *EndRouteSubscriber* descrita en la Sección 4.4.2.

4.4.4. StartRouteSubscriber

Este subscriptor se ha creado para detectar el inicio de un desplazamiento, nosotros hemos identificado esta situación cuando, tras haber estado un tiempo sin detectar nuevos puntos GPS, empezamos a detectar nuevos porque el usuario está en movimiento.

- **Consulta**

```
private final String  Query = "select a2 "
        + "from pattern [every a1 = EndRouteEvent -> a2 = BasicEvent]";
```

Figura 23

Esta consulta devuelve todos los eventos del tipo *BasicEvent* que suceden después de un *EndRouteEvent*. Como salida devolvemos el *BasicEvent* posterior.

- **Update**

```
public void update(Map<String, BasicEvent> eventMap) {

    BasicEvent event = (BasicEvent) eventMap.get("a2");

    StringBuilder sb = new StringBuilder();
    sb.append("---------------------------------");
    sb.append("\n- [Inicio Desplazamiento]: ");
    sb.append("\n latitud = " + event.getLatitud());
    sb.append("\n longitud = " + event.getLongitud());
    sb.append("\n speed = " + event.getSpeed());
    sb.append("\n time = " + event.getTimestamp().toString());
    sb.append("\n---------------------------------");
    System.out.println(sb);
    CSVReader.EPL.handle(new StartRouteEvent(event));
}
```

Figura 24

Esta operación *update* del subscriptor imprime por la pantalla de la consola las principales propiedades del suceso y envía un objeto del tipo *StartRouteEvent* a nuestro analizador CEP, *EPLEngine*.

4.4.5. StartRouteFirstSubscriber

Al igual que en el caso de la detención, al inicio de un desplazamiento no tenemos en cuenta una situación con la consulta que realizamos. Este caso es el primer Evento de nuestro análisis, que sin duda debemos interpretar como el inicio de un desplazamiento. Para lograr esto le pusimos dentro de la propiedad *flag* una cadena de texto, “inicio”, al primer evento dentro de nuestro archivo.csv.

- **Consulta**

```
private final String  Query = "select a1 "
        + " from BasicEvent as a1 where flag = \"inicio\" ";
```

Figura 25

Esta consulta devuelve todos los *BasicEvent* que contengan en la propiedad *flag* la cadena de texto “inicio”.

- **Update**

Como lo detectado en esta operación se trata de un *StartRouteEvent*, su operación *update* es la misma que la del subscriptor *StartRouteSubscriber* vista en la Sección 4.4.4.

4.4.6. RouteSubscriber

Este subscriptor se ocupa de detectar un desplazamiento. La consulta de este evento debe detectar una situación que transcurre a lo largo de un tiempo, no como en los demás eventos que suceden en un instante preciso. Hemos definido un desplazamiento como todos los eventos que transcurren en medio de un *StartRouteEvent* y *EndRouteEvent*. Por falta de recursos solo obtenemos estos dos últimos en esta consulta. Esta cuestión, relacionada con mejoras, es abordada más adelante en la Sección 5.

- **Consulta**

```
private final String  Query =
        "select a1, a2 "
        + "from pattern "
        + "[ every (a1 = StartRouteEvent -> a2 = EndRouteEvent) ] ";
```

Figura 26

Esta consulta se activa cada vez que después de un inicio de *StartRouteEvent* sucede un *EndRouteEvent*, sin importar que sucedan en medio otros eventos. Devuelve tanto el evento de inicio como el de fin de un desplazamiento.

- **Update**

```
public void update(Map<String, InterfaceEvent> eventMap) {

    StartRouteEvent event = (StartRouteEvent) eventMap.get("a1");
    EndRouteEvent fin = (EndRouteEvent) eventMap.get("a2");

    StringBuilder sb = new StringBuilder();
    sb.append("---------------------------------");
    sb.append("\n- [Desplazamiento inicio]: ");
    sb.append("\n latitud_ini = " + event.getLatitud());
    sb.append("\n longitud_ini = " + event.getLongitud());
    sb.append("\n speed_ini = " + event.getSpeed());
    sb.append("\n time_ini = " + event.getTimestamp().toString());
    sb.append("\n- [Desplazamiento fin]: ");
    sb.append("\n latitud_fin = " + fin.getLatitud());
    sb.append("\n longitud_fin = " + fin.getLongitud());
    sb.append("\n speed_fin = " + fin.getSpeed());
    sb.append("\n time_fin = " + fin.getTimestamp().toString());
    sb.append("\n---------------------------------");
    System.out.println(sb);

    CSVReader.EPL.handle(new RouteEvent(event,fin));
}
```

Figura 27

La función *update*, aparte de imprimir por pantalla las principales propiedades de los *StartRouteEvent* y *EndRouteEvent*, envía al objeto de la clase *EPLEngine* un evento complejo del tipo *RouteEvent*.

4.4.7. RouteTrustlySubscriber

Nos dimos cuenta de que nuestro anterior *RouteSubscriber* generaba demasiadas rutas falsas. Al encontramos dentro de un techado, nuestro GPS puede perder la señal y obtener falsos movimientos. Estos falsos positivos no suelen durar mucho tiempo, por lo tanto, añadimos este subscriptor para descartar los desplazamientos falsos. Básicamente, los desplazamientos que duren pocos segundos no los consideramos como desplazamiento.

- **Consulta**

```
private final String  Query =
    "select a1 "
  + "from RouteEvent a1  "
  + "where (timestamp2().getTime() - timestamp().getTime())/1000 > 120";
```

Figura 28

Esta consulta devuelve todos los eventos del tipo *RouteEvent* que transcurren en un tiempo superior a 120 segundos.

- **Update**

```
public void update(Map<String, InterfaceEvent> eventMap) {

    RouteEvent event = (RouteEvent) eventMap.get("a1");

    StringBuilder sb = new StringBuilder();
    sb.append("---------------------------------");
    sb.append("---------------------------------");
    sb.append("\n- [Desplazamiento Trustly inicio]: ");
    sb.append("\n latitud_ini = " + event.getLatitud());
    sb.append("\n longitud_ini = " + event.getLongitud());
    sb.append("\n speed_ini = " + event.getSpeed());
    sb.append("\n time_ini = " + event.getTimestamp().toString());
    sb.append("\n- [Desplazamiento fin]: ");
    sb.append("\n latitud_fin = " + event.getLatitud2());
    sb.append("\n longitud_fin = " + event.getLongitud2());
    sb.append("\n speed_fin = " + event.getSpeed2());
    sb.append("\n time_fin = " + event.getTimestamp2().toString());
    sb.append("\n---------------------------------");
    System.out.println(sb);
    //todo: named window to speed analysis
    CSVReader.CoordinatesList.putEventMap(Color.WHITE, event.getLatitud(), event.getLongitud());
    CSVReader.CoordinatesList.putEventMap(Color.BLACK, event.getLatitud2(), event.getLongitud2());
}
```

Figura 29

En esta función devolvemos por la pantalla del terminal las propiedades principales del inicio y el fin del desplazamiento. Además, son guardadas las coordenadas latitud y longitud del inicio y el fin de los desplazamientos detectados. Asignamos el color blanco para los inicios y el negro para las detenciones.

4.4.8. DirectionSubscriber

Este subscriptor fue creado para calcular la dirección de un *BasicEvent*. Esta dirección se calcula teniendo en cuenta el próximo evento que sucede para generar un vector dirección en relación a la latitud y longitud de estos.

- **Consulta**

```
private final String  Query = "select a1, a2 "
        + "from pattern [ every a1 = BasicEvent ->  a2 = BasicEvent ]";
```

Figura 30

Esta consulta devuelve todos los eventos consecutivos en parejas de dos con la finalidad de poder hallar la dirección del primero.

- **Update**

```
public void update(Map<String, BasicEvent> eventMap) {

    BasicEvent a1 = (BasicEvent) eventMap.get("a1");
    BasicEvent a2 = (BasicEvent) eventMap.get("a2");
    float a, b, c;

    a = a1.getLatitud()-a2.getLatitud();
    a = a*111111;
    b = a1.getLongitud()-a2.getLongitud();
    b = (float) ( b * Math.cos(a2.getLatitud()*Math.PI/180)*111111 );
    c = (float) Math.sqrt(a*a + b*b);
    Vec2f result = new Vec2f(a/c, b/c);

    StringBuilder sb = new StringBuilder();
    sb.append("\n- [Incremento en]: ");
    sb.append("\n direccion latitud = " + a);
    sb.append("\n direccion longitud  = " + b);
    CSVReader.EPL.handle(new DirectionEvent(a1,result));
}
```

Figura 31

En esta función calculamos un vector dirección en función a la latitud y longitud de los dos eventos que hemos obtenido de la consulta. Para hallar esta dirección hacemos una transformada aproximada, basándonos en el radio de la tierra, que son unos 40,000km, dividido entre 360 grados, obtenemos que cada grado de latitud son 111,111metros. Además, el vector dirección calculado lo convertimos en un vector unitario. Finalmente, enviamos a *EPLEngine* un evento del tipo *DirectionEvent* el cual posee las características del primer evento junto a la dirección, representada por el vector, que hemos calculado previamente.

4.4.9 ChangeDirectionSubscriber

Esta consulta determina si hubo un cambio de dirección comparado dos direcciones consecutivas. Si la diferencia de ambas es mayor a 36º determinamos que hubo un cambio de dirección.

- **Consulta**

```
private final String  Query =
        "select a1, a2 "
      + "from pattern [ every a1 = DirectionEvent -> a2 = DirectionEvent ]";
```

Figura 32

Consulta que devuelve dos eventos consecutivos del tipo *DirectionEvent*, el cual posee un vector con la dirección a la que esta apuntando.

- **Update**

```
public void update(Map<String, DirectionEvent> eventMap) {

    DirectionEvent a1 = (DirectionEvent) eventMap.get("a1");
    DirectionEvent a2 = (DirectionEvent) eventMap.get("a2");
    float angulo;

    angulo = (float) Math.acos( a1.getDirection().x*a2.getDirection().x+a1.getDirection().y*a2.getDirection().y );
    boolean hayCambio = Math.abs(angulo)  >  Math.PI/5;

    StringBuilder sb = new StringBuilder();
    sb.append("\n- [Cambio de direccion]: ");
    sb.append("\n- [Cambio de direccion]: ");
    sb.append("\n- Coordenadas de cambio... ");
    sb.append("\n- Latitud = " + a2.getLatitud() );
    sb.append("\n- Longitud = " + a2.getLongitud() );
    sb.append("\n- Tiempo = " + a2.getTimestamp().toString());
    sb.append("\n- Pi = " + Math.PI/5);
    sb.append("\n- Tiene que ser mayor = " + angulo);

    if(hayCambio){
        System.out.println(sb);
        CSVReader.CoordinatesList.putEventMap(Color.green, a2.getLatitud(), a2.getLongitud());
    }
}
```

Figura 33

En esta función calculamos la diferencia de ángulos que poseen ambas direcciones. Hemos simplificado mucho los cálculos ya que los vectores de ambas direcciones son unitarios. Si se da que el ángulo de ambas direcciones es mayor a 36º, consideramos que hubo un cambio de dirección y almacenamos las coordenadas GPS en nuestra variable junto con el color verde color para después identificar su situación en el mapa.

4.5. Eventos

Cada evento debe simbolizar una situación del mundo real. Los eventos son las clases que usamos para almacenar la información con la que nuestro sistema tiene que trabajar. Solo hemos implementado en el sistema los eventos que forman parte de algún análisis, ya que no es necesario implementar las demás clases. Es decir, *RouteTrustlyEvent* y *ChangeDirectionEvent* no están implementadas.

4.5.1. BasicEvent

Esta clase es el evento básico que generamos con el fichero producido por "*SmartUs*". La función principal de este es almacenar de forma ordenada todos los datos importantes procesados por nuestra clase *CSVReader*, toda esta información es recogida del exterior por los sensores de nuestro móvil. A continuación, en la Figura 34 se muestran los datos guardados.

```
private float longitud;
private float latitud;
private float speed;
private Calendar time;
private Date timestamp;
private String flag;
```

Figura 34: Propiedades BasicEvent

Este evento dispone de un método *CEPOnlineString* que devuelve un texto del evento con todos sus datos en el formato especificado para su uso en Esper online, fíjese en la Figura 35.

```
public String CEPOnlineString()
{
    return "BasicEvent ={ timestamp = \""+time.getTime()+"\", "
     + "latitud = "+latitud+", longitud = "+longitud+",Speed = "+speed+" }";
}
```

Figura 35: Función CEPOnlineString

Por último, cabe mencionar que *BasicEvent* es el evento semilla del que parten el resto de los eventos que nuestro análisis. Por lo tanto, el resto de evento que vamos a nombrar aparecen en nuestro sistema a partir del análisis inicial del *BasicEvent*.

4.5.2. InterfaceEvent

Hemos creado una interfaz para poder agrupar todos los eventos complejos con las mismas funcionalidades y que puedan ser manejados por igual en algunas funciones, como por ejemplo la función *handle* mostrada en la Figura 17.

4.5.3. ComplexEvent

Todos los eventos complejos poseen bastantes atributos comunes los cuales heredan de esta clase. Además, poseen las variables comunes que mostramos en la siguiente Figura 36.

```
private float longitud;
private float latitud;
private float speed;
private Calendar time;
private Date timestamp;
private String flag;
```

Figura 36: Propiedades ComplexEvent

También, consideramos como funciones comunes todos los *get* y *set* de las variables mostradas y un constructor.

4.5.4. EndRouteEvent

Este evento se identifica con el momento final de un desplazamiento, cuando el individuo se detiene. Los subscriptores que generan este tipo de eventos son *EndRouteSubscriber* (Sección 4.4.2) y *EndRouteLastSubscriber* (Sección 4.4.3).

4.5.5. StartRouteEvent

Este evento se identifica con el momento inicial de un desplazamiento, cuando el individuo empieza a desplazarse después haber estado detenido. Los subscriptores que generan este tipo de eventos son *StartRouteSubscriber* (Sección 4.4.4) y *StartRouteFirstSubscriber* (Sección 4.4.5).

4.5.6. RouteEvent

Este evento simboliza un desplazamiento. Se trata situación que ocurre a lo largo de un tiempo, y está delimitado por un momento inicial y un momento final. Para almacenar toda la información relevante de esta situación hemos tenido que crear más variables para representar el momento inicial. Como se aprecia en la Figura 37 este evento es como si estuviera compuesto por dos *ComplexEvent*, ya que tiene la variable duplicadas para almacenar el momento de detención.

```
private float longitud2;
private float latitud2;
private float speed2;
private Calendar time2;
private Date timestamp2;
private String flag2;
```

Figura 37: Propiedades RouteEvent

El subscriptor que genera este tipo de evento es *RouteSubscriber*, véase en la Sección 4.4.6.

4.5.7. DirectionEvent

Este evento nos sirve para almacenar la dirección que lleva un *BasicEvent*. Además de todas las variables propias de un *ComplexEvent* posee una variable vectorial de dos dimensiones en la cual se almacena la dirección. El subscriptor que genera este tipo de evento es *DirectionSubscriber*, descrito en la Sección 4.4.8. A continuación, en la Figura 38 mostramos la variable vectorial que hemos mencionado anteriormente.

```
public class DirectionEvent  extends ComplexEvent {

    Vec2f direction;

    public DirectionEvent(BasicEvent e,Vec2f vectorDirection) {
        super(e);
        this.direction = vectorDirection;
    }

    public Vec2f getDirection() {
        return direction;
    }


}
```

Figura 38: Función DirectionEvent

4.6. Mapas

Como hemos mencionado en nuestra memoria de desarrollo, era necesario implementar un mapa con el fin de poder interpretar los datos con mayor facilidad. En esta sección explicamos todas las clases que intervienen en el dibujado del mapa en nuestro sistema.

4.6.1. MapCreator

Extiende de la interfaz JFrame, la cual, sirve para crear ventana con las funcionalidades básicas de cerrar, minimizar y extender. Para crear esta ventana tenemos que realizar unas configuraciones básicas, mostradas en la Figura 39, de cómo crear ventanas en Java.

```
GridBagLayout gbl = new GridBagLayout();
GridBagConstraints gbc = new GridBagConstraints();
this.setTitle("Route Analysis");
this.setLayout(gbl);
setExtendedState(JFrame.MAXIMIZED_BOTH);
gbc.fill = GridBagConstraints.BOTH;
gbc.gridwidth = 1;
gbc.weighty = 1;
gbc.weightx = 1;
gbc.gridx = 0;
gbc.gridy = 0;
```

Figura 39: Configuracón Mapa

Después debemos crear un objeto JMapViewer, que se trata del mapa en sí que queremos pintar sobre la ventana que vamos a desplegar. A este le añadimos su controlador por defecto y lo centramos y ampliamos sobre la ciudad de Málaga ya que es nuestro principal escenario de pruebas.

```
map = new JMapViewer();
DefaultMapController mapController = new DefaultMapController(map);
mapController.setMovementMouseButton(MouseEvent.BUTTON1);
map.setSize(800, 800);
GotoMalaga();
```

Figura 40: Controladores Mapa

A continuación, debemos crear la vista y añadirle funcionalidades básicas a nuestra ventana.

```
add(map, gbc);
pack();
setVisible(true);
overlayI = new BufferedImage(getWidth(), getHeight(),
                BufferedImage.TYPE_INT_ARGB);

addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
                System.exit(0);
        }
});
```

Figura 41: Controladores Ventana

Finalmente, usamos el componente *CoordenateList* de nuestro sistema para dibujar en el mapa los eventos registrados en nuestro análisis por medio de la función *EditMap* que mostramos en la Figura 42.

```
public void EditMap(CoordinateList coordinateEvents){
    for(MapMarkerDot Point: coordinateEvents.getEventPos()){
        map.addMapMarker(Point);
    }
}
```

Figura 42: Función EditMap

4.6.2. CoordenateList

Es una clase que hemos creado con la funcionalidad de ir guardando todas las coordenadas de eventos interesantes que hemos detectado en nuestro análisis con el objetivo de mostrarlas en el mapa anterior.

Esta clase es usada por los subscriptores en algunos casos cuando detectamos un evento interesante que nos interesa pintar en nuestro mapa. A continuación, mostramos una leyenda en la Figura 43 con los diferentes tipos de puntos que dibujamos en nuestro mapa y que simbolizan cada uno, además del subscriptor que lo ha generado.

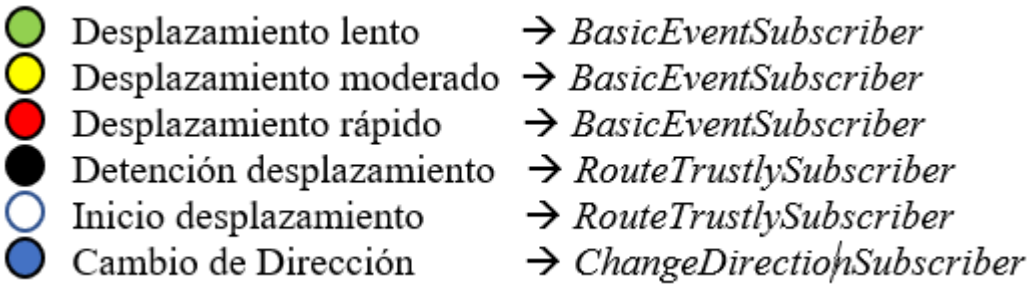

Figura 43: Colores de los eventos en el Mapa

Básicamente, esta clase solo contiene una lista de objetos puntos con manejadores para usarla de forma más sencilla. En la próxima Figura 44, mostramos toda la estructura de esta clase.

```
public  class CoordinateList {
    List<MapMarkerDot> DotEventList;

    public CoordinateList(){
         DotEventList = new  ArrayList<MapMarkerDot>();
    }

    public List<MapMarkerDot> getDotEventList() {
        return DotEventList;
    }

    public void putDot(Color color,double latitud,double longitud){
        MapMarkerDot a = new MapMarkerDot(Color.BLACK, latitud, longitud);
        a.setBackColor(color);
        DotEventList.add(a);
    }

}
```

Figura 44: Función CoordinateList

5. Análisis de Resultados

En esta sección vamos a analizar los casos de pruebas que hemos hecho y las situaciones especiales con las que nos hemos encontrado realizando las pruebas del sistema que se ha desarrollado. También mencionaremos las principales fortalezas y debilidades de nuestro proyecto para justificar en la Sección 6 cuales son las líneas futuras de este proyecto y como se debe orientar su desarrollo.

5.1. Casos de Prueba

Hemos establecido los casos de prueba según los diferentes objetivos propuestos en la introducción de esta memoria en la Sección 1.3. Además, estas pruebas se muestran desde diferentes medios de transporte, principalmente en desplazamientos en coche, autobús urbano, metro y a pie.

Estos casos de pruebas no solo nos muestran si las consultas funcionan adecuadamente, sino que también nos ayudan a inspirarnos para crear nuevas reglas. Un ejemplo es la implementación de nuestro subscriptor *RouteTrustlySubscriber* de la Sección 4.4.7. Hay muchas mejoras que nos gustaría haber añadido y proponemos dentro de esta sección en los diferentes casos de prueba.

Empezaremos por los casos más simples primero, y en adelante se irán mostrando escenarios de pruebas cada vez más complejos.

5.1.1. Caso de Prueba 1: Desplazamiento simple andando "*caminando.csv*"

Figura 45: caminado.csv

Este es el caso de prueba más simple de todos, un único desplazamiento andando al aire libre. En este se muestra como el usuario de la aplicación se desplaza desde el centro de Málaga hacia el Parque San Antonio. Como se aprecia, todos los puntos son de color verde ya que en ningún momento se desplaza a una alta velocidad (ya que va andando). Al caminar al aire libre en ningún momento se pierde la señal GPS, por lo tanto, no hay ninguna incidencia dentro este análisis

5.1.2. Caso de Prueba 2: Desplazamiento simple en bicicleta "bicicanasta.csv"

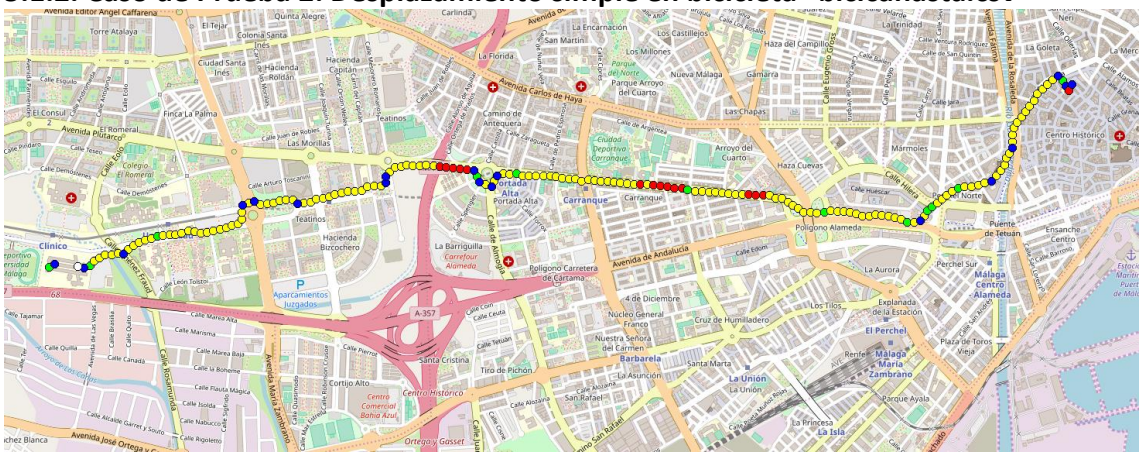

Figura 46: bicicleta.csv

En esta ocasión el usuario se desplaza usando una bicicleta y tampoco hace ninguna parada larga. El trayecto transcurre desde nuestra escuela de informática hasta el centro de Málaga. Como se puede apreciar dentro de todo este desplazamiento se alcanzan todo tipo de velocidades (verde amarilla y roja). Se puede apreciar dentro de este desplazamiento que la mayoría de *BasicEvent* dibujados en el son de color amarillo, por lo tanto, a partir de este análisis podríamos deducir que si la mayoría de los puntos son amarillos el usuario se ha desplazado en bicicleta.

Cabe mencionar que hay un evento interesante que no detectamos dentro del desplazamiento del usuario. Se trata de que el usuario hizo una pequeña parada para comprar pan en una panadería del trayecto. De aquí nace una idea para implementar en futuras iteraciones de este proyecto, el concepto de *Microparada*. Una *Microparada* es una parada de poco tiempo dentro de un desplazamiento. Esto nos ayudaría a detectar si el usuario estuvo durante un breve tiempo en un lugar o de si este se detuvo en un semáforo. Para poder saber esto sería necesario también implantar Geolocalización inversa en el proyecto.

5.1.3. Caso de Prueba 3: desplazamiento en coche "*Ikea.csv*"

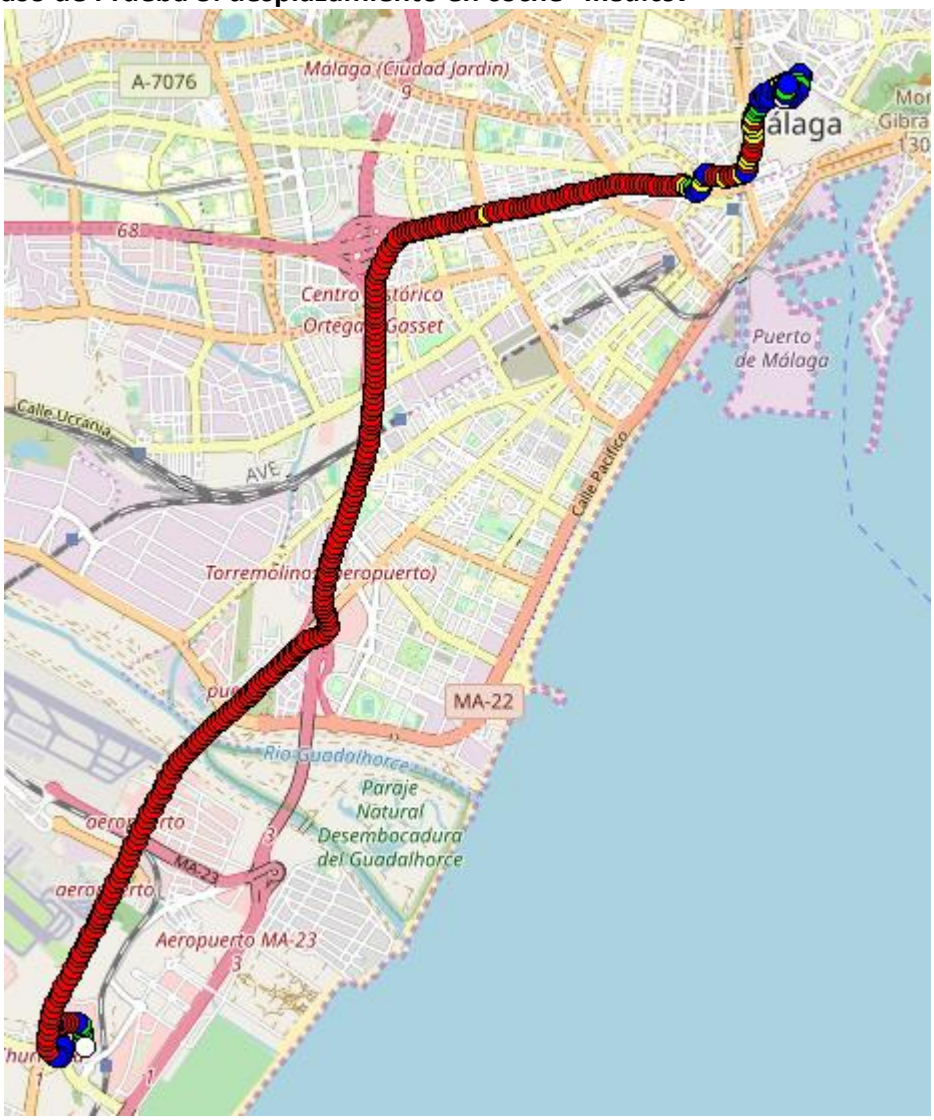

Figura 47: Ikea.csv

Este desplazamiento empieza desde el Ikea y acaba en el centro de Málaga. Como se puede apreciar la mayoría de los *BasicEvent* son de color rojo ya que lleva una velocidad alta. Para poder analizar mejor este caso de prueba vamos a tomar otras dos capturas del inicio y del fin del trayecto que es donde encontramos más casos particulares.

Figura 48: Ikea.csv inicio desplazamiento

En el inicio de este desplazamiento el usuario sale del Ikea andando en busca de su coche hasta que se monta en el para continuar al centro de la ciudad. Véase que en la Figura 48 se aprecia perfectamente que al principio los *BasicEvent* son de color verde ya que el usuario se encuentra andando y al coger su vehículo estos cambian de color rojo.

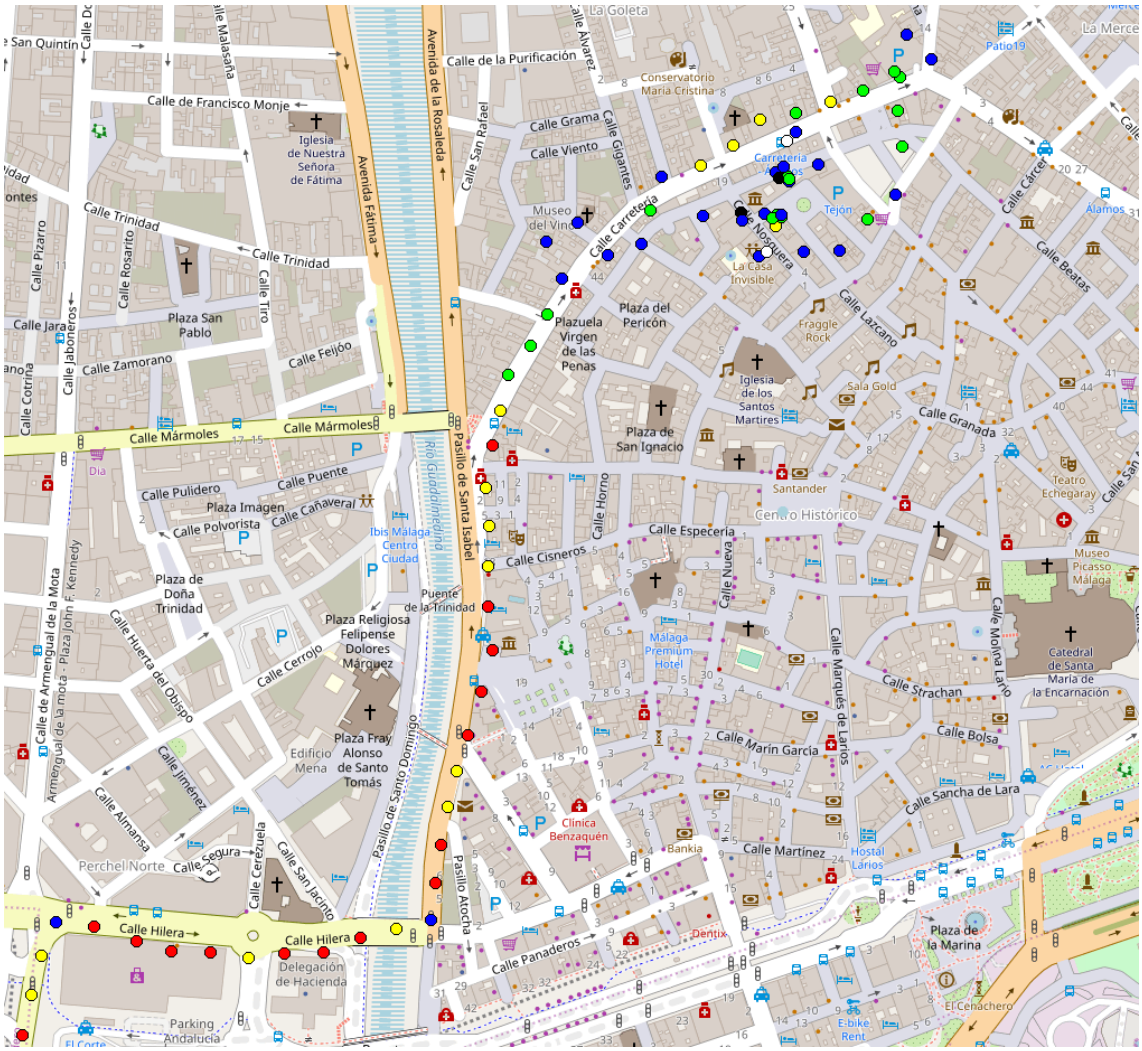

Figura 49: Ikea.csv fin desplamiento

Al final del trayecto vemos como los puntos vuelven a ser verdes, pero no podemos saber si es porque el usuario se bajó del coche o había mucho tráfico por la zona y tenía que desplazarse lentamente. Esta situación se estudiará en futuras iteraciones.

5.1.4. Caso de Prueba 4: Desplazamiento en autobús y andando "*caminabus.csv*"

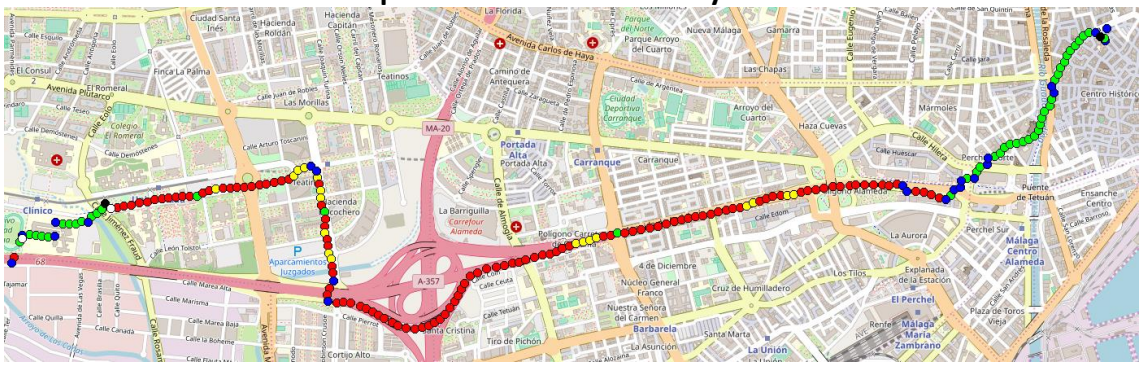

Figura 50: caminabus.csv

Esta es el primer análisis en el que podemos apreciar una parada intermedia en dentro del conjunto de datos analizado. El usuario se desplaza desde nuestra escuela hasta una parada de autobús, ahí espera a el autobús para montarse en él y bajarse cerca de El

Corte Inglés y seguir andando hasta el centro de Málaga. Nuestro sistema ha sido capaz de detectar dos desplazamientos, ya que el usuario se detuvo en la parada de autobús.

Al inicio del primer desplazamiento hay una pequeña perdida de señal del GPS. Esto ocurre en varios casos, y se debe a que se acaba de activar el GPS. Esta incidencia se tendrá en cuenta para futuras iteraciones del proyecto. Una solución podría ser descartar los primeros datos recopilados de cada análisis o desde "*SmartUs*" no enviar datos hasta que se sepa que el GPS está estable.

Nótese que durante el segundo desplazamiento, durante el desplazamiento en autobús, la mayoría de *BasicEvent* son de color rojo, pero estos pasan a ser de color verde porque el usuario se bajó la parada. De aquí se nos ocurrió una implementación para el futuro que sería detectar el cambio de velocidad continuado del usuario durante un desplazamiento con el fin de detectar este evento de cambio de medio de transporte.

5.1.5. Caso de Prueba 5: análisis usando el metro "metro.csv"

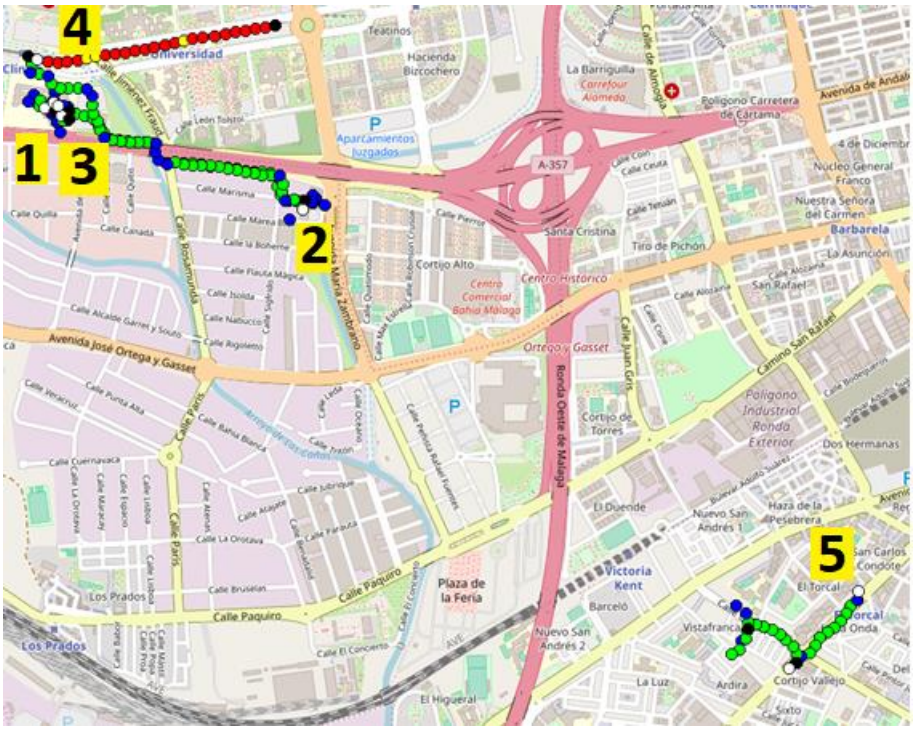

Figura 51: metro.csv

Vamos a enumerar el recorrido de la Figura 51 para que sea más fácil de analizar. Partimos de la escuela (1) hasta el punto (2) volvemos a la escuela (3), vamos a la parada de metro (4) subiéndonos en él y acabamos en la parada de metro del Torcal (5).

Vamos a analizar primero las paradas (1) (2) y (3), en las cuales, el GPS pierde la señal por encontrarnos en un sitio interior. En el caso (2) no es ningún problema ya que las pérdidas son durante un pequeño momento y gracias al subscriptor *RouteTrustlySubscriber,* de la Sección 4.4.7, descartamos esas pérdidas de GPS como no desplazamientos. Pero en el caso (1) y (3) la señal GPS se pierde durante intervalos de tiempo muy lagos generando varios puntos de parada como se puede apreciar en la

Figura 51. Estas pérdidas de GPS ocasionan problemas para nuestro análisis, ya que ocasionan desplazamientos no reales. Como solución para futuras iteraciones se aconsejan dos caminos diferentes. Crear un conjunto de consultas de sean capaces de descartar estas pérdidas de GPS; O dentro de "*SmartUs*" encontrar un método para solo recopilar trazas GPS cuando la señal sea buena, ya que la señal es mala cuando el usuario se encuentra en un interior.

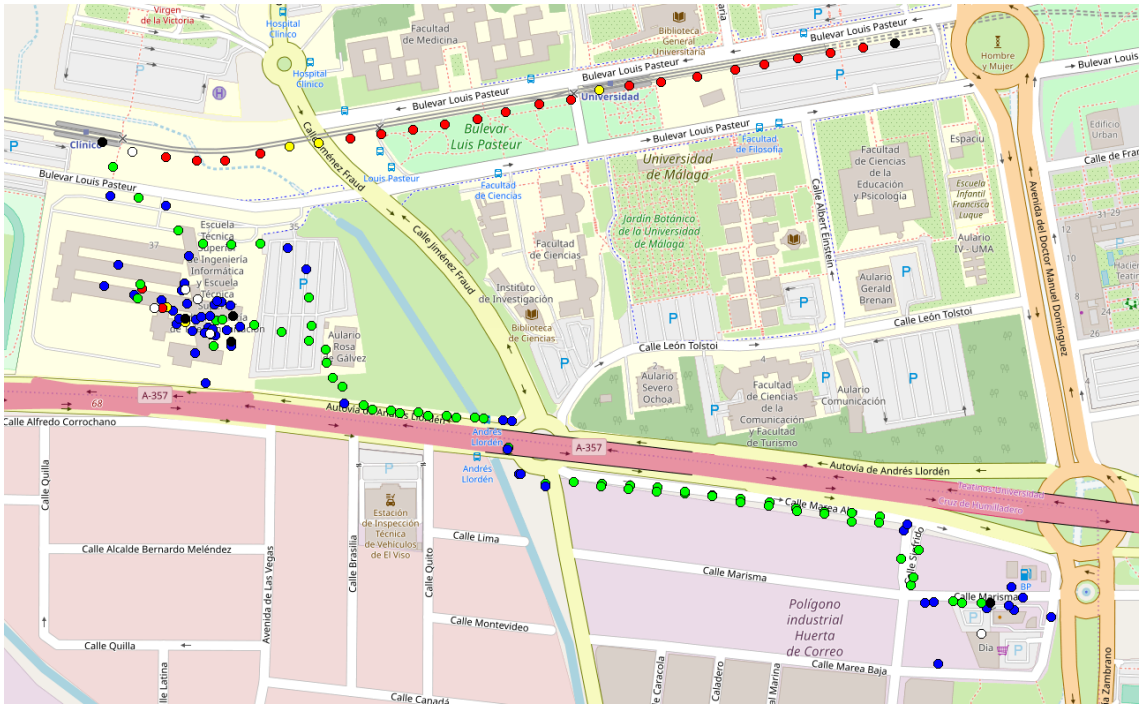

Figura 52: metro.csv inicio

Por último, en los casos (4) y (5) se tratan de un viaje en metro. Este tipo de viaje no son detectados aun por nuestro sistema. Proponemos, para una futura iteración, una solución para detectar este tipo de viajes en los que no hay ninguna señal GPS. La solución sería analizar donde acaba un desplazamiento y empieza el próximo desplazamiento. Si ambos puntos se tratan de una parada de metro, podemos deducir que el usuario viajo en metro.

5.1.6. Caso de Prueba 6: Antes y después de RouteTrustlySubscriber

Figura 53: Antes de TrustlyRouteSubscriber

Figura 54: Después de TruslyRouteSubscriber

Este caso de prueba muestra los análisis la del subscriptor *RouteTrustlySubscriber* antes (Figura 53) y después (Figura 54) de su implementación. La principal funcionalidad de este suscriptor es filtrar todos los desplazamientos falsos que se crean por la pérdida de señal GPS durante un pequeño plazo de tiempo. El principal problema de este ajuste se muestra en la interrogación "?" de la Figura 54 donde perdemos un desplazamiento real que ocurre en un pequeño intervalo de tiempo. Por un lado, eso no es un gran problema ya que la finalidad de este estudio va centrada en desplazamiento largos.

5.1.7. Caso de Prueba 7: Desplazamiento largo en coche “MalagaCaceres.csv”

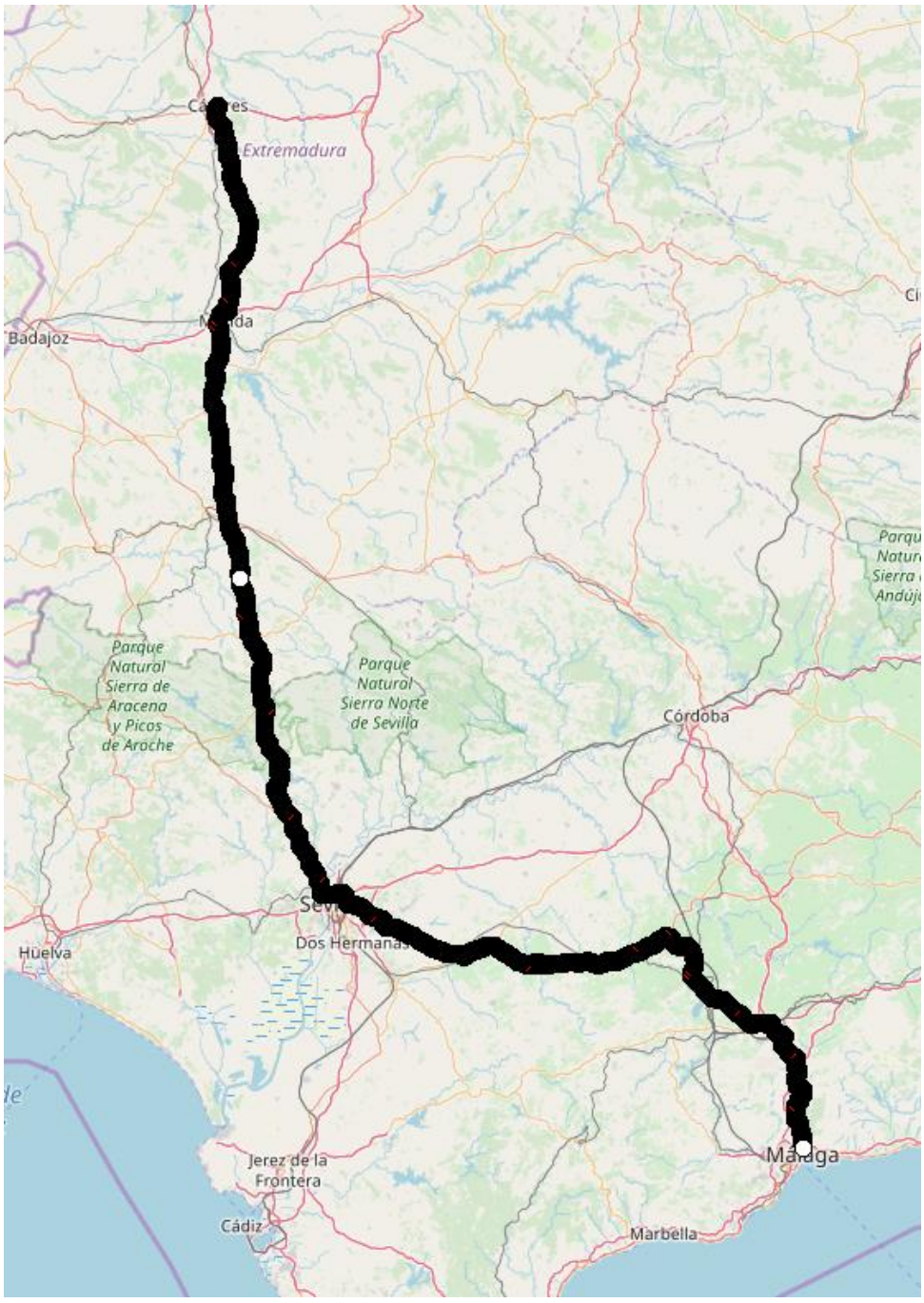

Figura 55: MalagaCaceres.csv

Este caso de prueba se analiza un viaje en coche desde Málaga hasta Cáceres. En la figura X debido a la agrupación excesiva de puntos se ven negros al estar solapados, pero si ampliamos la imagen se pueden ver perfectamente. Y a continuación, en la siguiente Figura 55 se muestra que en medio de este desplazamiento se detecta con éxito una parada a mitad de trayecto dentro de un restaurante (1). Además, dentro de la ciudad de Cáceres se muestra cómo se llega y a continuación se recorre algunos tramos de la ciudad a pie (2).

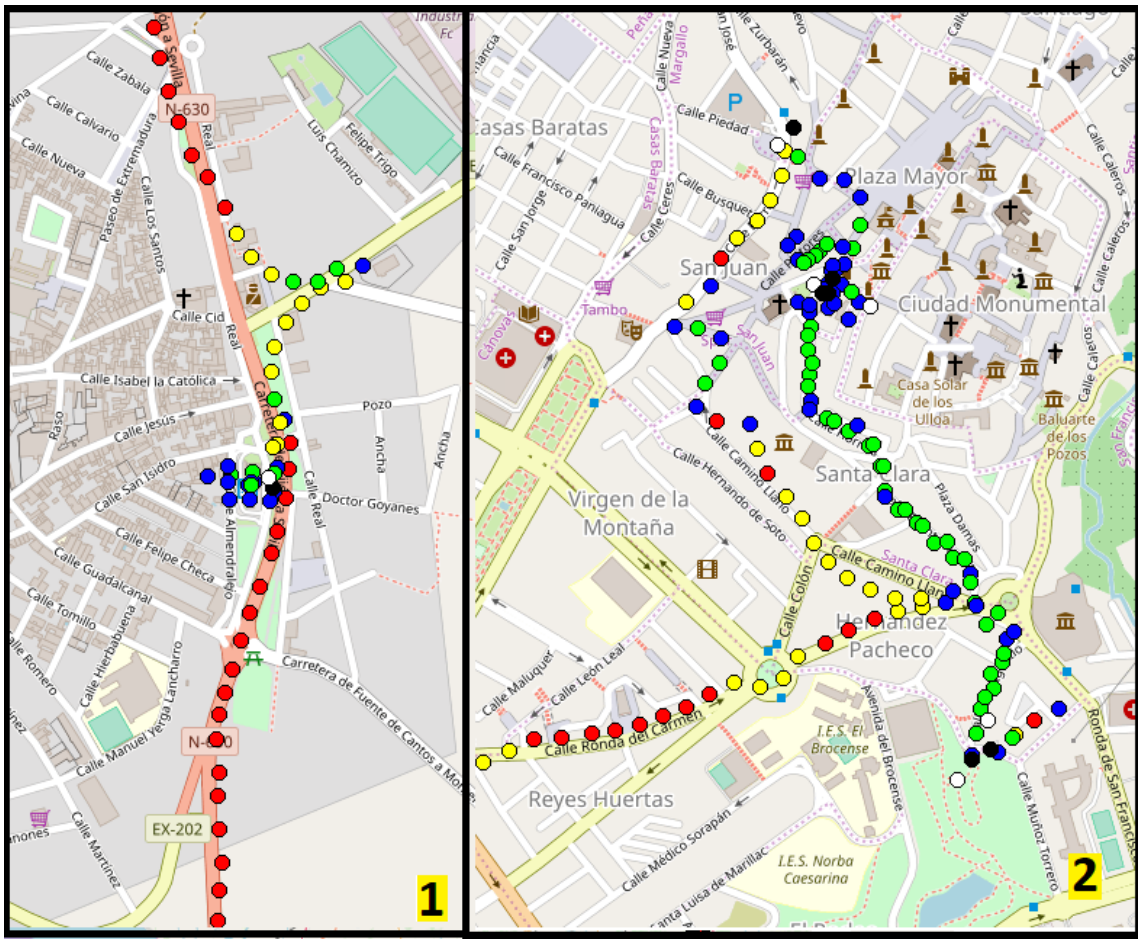

Figura 56:MalagaCeceres parada y llegada

No hemos hablado en los casos de prueba del subscriptor *ChangeDirectionSubscriber* de la Sección 4.4.9 porque por lo general se comportaba adecuadamente dentro de los análisis mostrando resultados muy fiables de cuando hay un cambio de dirección en el recorrido del usuario. Pero en este largo trayecto en coche se han detectado muchos falsos positivos como se aprecia en la Figura 56. El principal problema puede deberse a la aproximación que se hace para convertir las coordinas latitud longitud en un sistema cartesiano plano. Este problema será revisado en futuras iteraciones de este proyecto.

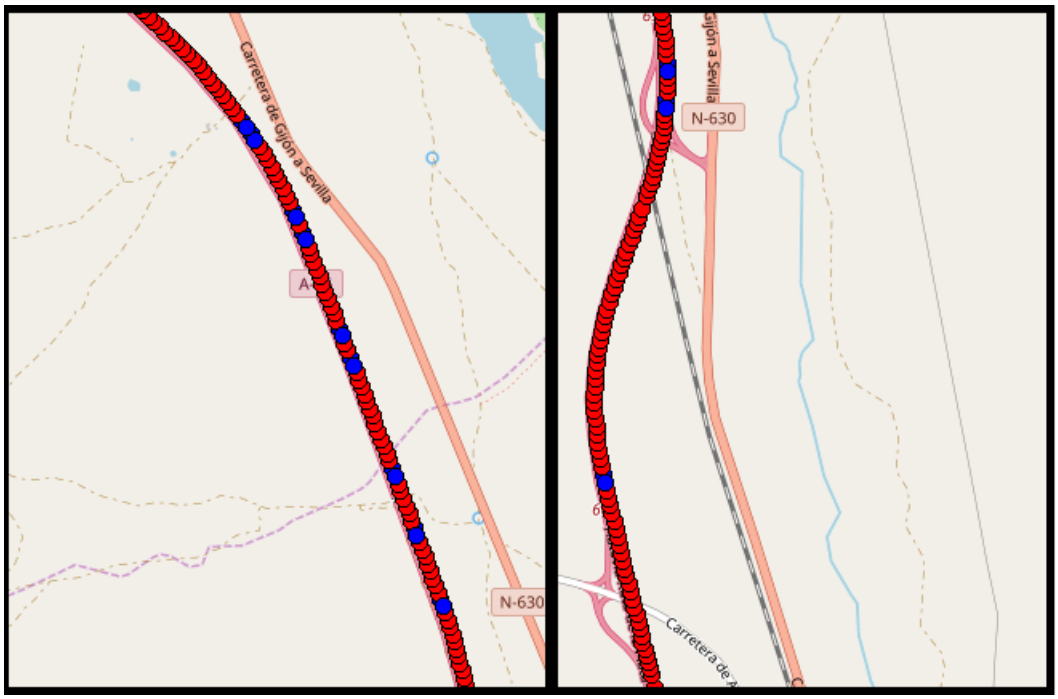

Figura 57: ChangeDirectionSubscriber falsos positivos

5.2. Fortalezas

En este apartado se hace hincapié de los puntos fuertes de los análisis hechos dentro de los casos de prueba del apartado anterior. Se muestra como estas fortalezas se ciñen a los principales objetivos planteados dentro de esta memoria. A continuación, mostramos estas fortalezas:

- El principal objetivo de detectar los desplazamientos que se realizan dentro de los datos analizados ha sido exitosamente alcanzado, sobre todo cuando estos datos son recopilados en un espacio exterior.
- Los datos analizados son mostrados en un mapa donde son mucho más fáciles de evaluar para comprobar que las consultas realizadas en cada análisis corresponden con una situación real. Además, mostrarlos en el mapa nos ha ayudado a pensar en nuevas mejoras que podemos añadir en nuestros análisis.
- Se ha podido analizar correctamente como se ven en las imágenes anteriores que efectivamente cuando las trazas son dibujadas en rojo se trata de un transporte motorizado, mientras que las amarillas son de un desplazamiento en bicicleta y las verdes representan un desplazamiento caminando.
- Nuestro analizador, a excepción dentro de un viaje largo en carretera, es capaz de detectar el momento en el que el usuario efectúa un cambio de dirección dentro de su desplazamiento. Los resultados obtenidos dentro de la ciudad y viajes cortos detectan con muy buena calidad los momentos en los que se da esta situación.

5.3. Debilidades

Ahora a modo de resumen relataremos las principales debilidades que se comentan en el apartado de casos de prueba. Todas están debilidades irán junto con una solución que será aplicada en futuras iteraciones de este proyecto. Muchas de estas debilidades son de escenarios particulares que se tuvieron en cuenta a la hora de desarrollar este proyecto. Estas son las debilidades:

- Periodos largos con pérdidas de la señal GPS del dispositivo móvil. Estos periodos generan falsas rutas en nuestros análisis. Podemos plantear la solución de este problema de dos formas, una de ellas sería crear consultas capaces de detectar efectivamente estas malas señales de GPS y descartarlas de los análisis, y la otra sería obviarlas encontrando un método para detener la recolección dentro de la aplicación "*SmartUs*" cuando esta detecta que las señales GPS son malas.
- En el mapa no deberían mostrarse todos los *BasicEvent*. Solo deberían aparecer los identificados dentro de una ruta real en el subscriptor *RouteTrustlySubscriber*. De esta forma conseguiremos unos resultados más limpios eliminando todas las malas señales del GPS. Para hacer esto se debería implementar un subscriptor capaz de detectar todos los *BasicEvent* sucedidos dentro de un desplazamiento real, e identificar estos como un evento diferente.
- Refactorizar el subscriptor ChangeDirectionSubscriber para evitar los falsos positivos que puede detectar esta consulta. Para ello proponemos crear un método robusto para convertir la latitud y longitud en coordenadas cartesianas de un plano.

6. Conclusiones y Proyectos Futuros

Este proyecto refleja un estudio, no solo del análisis de los desplazamientos del usuario, sino que también hacemos un estudio sobre una tecnología con muchas utilidades como es CEP. Dentro de este proyecto queda reflejado como se debe implementar CEP en cualquier estudio de cualquier materia. Este proyecto puede servir de base para cualquier persona que esté interesado en conocer más sobre esta potente herramienta de análisis en tiempo real.

Dentro de nuestro estudio de desplazamientos hemos logrado con éxito la recopilación de los distintos desplazamientos que realiza el usuario de la aplicación a lo largo de su rutina. El único inconveniente que ha tenido nuestro análisis ha sido a la hora de analizar trayectos con mala señal GPS. Hemos podido evitar estos problemas en el caso de que esta pérdida de señal suceda en periodos breves de tiempo, pero cuando estas pérdidas de señal son mayores a 2 minutos nuestro analizador crea desplazamientos ficticios.

Tras de haber establecido las nociones básicas de cómo usar CEP en desplazamientos, las líneas futuras de este proyecto deben ir dedicadas a mejorar el funcionamiento de las consultas EPL dentro de los análisis que se hacen. A continuación, mencionamos las principales mejoras que deben implantarse en este proyecto en base a los casos de prueba que hemos efectuado en la Sección 5.1:

- Identificar los *BasicEvent* dentro de un desplazamiento. La idea de esto es crear un nuevo evento de los *BasicEvent* que cumplan este requisito con el fin de hacer análisis más centrados en las ocurrencias sucedidas en los desplazamientos reales ya filtrados.
- Identificar el medio de transporte en dentro de los desplazamientos. El objetivo de este análisis es reconocer el momento en el que el usuario cambio de medio de transporte ya que sucedió un cambio drástico de velocidad.
- Implantar Geolocalización inversa. La geolocalización inversa consiste en obtener por medio de unas coordenadas latitud-longitud algo que se encuentre dentro de esa posición, por ejemplo, el nombre de la calle o el edificio más cercano. Esto se implantará con el fin de saber si el usuario se detuvo en una parada de autobús, o conseguir el itinerario de calles dentro de un desplazamiento.

- Evento de *Microparada*. Este evento tiene la finalidad de obtener todos los lugares en los que nuestro usuario se detuvo durante un pequeño tiempo dentro de su desplazamiento. Esto nos ayudará a recopilar información como los semáforos con los que se encuentra dentro de un desplazamiento, o si el usuario entró a algún establecimiento durante un breve periodo de tiempo.
- Crear un identificador que sea capaz de deducir cuando el usuario uso el metro para transportarse por medio de la perdida de señal que sucede. Actualmente este evento sucede cuando un fin de desplazamiento e inicio del próximo suceden en una parada de metro diferente.
- Descartar dentro de nuestros análisis los datos recopilados con una mala señal GPS. Esto se puede solucionar creando una consulta capaz de filtrar estos malos datos; O dentro de la aplicación "*SmartUs*" detener el envío de datos cuando la señal GPS es mala.
- Mejorar el cálculo de cambio de dirección usando un método fiable de conversión de coordenadas latitud-longitud en unas coordenadas dentro de un sistema cartesiano plano. Ejemplo web (http://www.ign.es/wcts-app/)

Siguiendo las líneas del TFM de "*SmartUs*", después de realizar esta serie de mejoras planteadas, el próximo paso sería transportar todo este sistema dentro de un dispositivo móvil. De esta forma los datos y los análisis efectuados quedarían dentro del teléfono móvil haciendo al usuario dueño de su propia información. Para finalizar esta línea de proyecto finalmente habría que usar los datos abiertos de Málaga para contrastar los análisis efectuados por nuestra herramienta CEP, y así, aconsejar que medio de transporte es más conveniente para cada desplazamiento efectuado.

7. Referencias

[1] J. Guillen, J. Miranda, J. Berrocal, J. Garcia-Alonso, J. M. Murillo, C. Canal., «“People as a Service: A Mobile-centric Model for Providing Collective Sociological Profiles”,» *Software, IEEE,* pp. 48-59, 2014..

[2] J. Berrocal, J. Garcia-Alonso, C. Canal, J.M. Murillo, «Situational-context:aunifiedviewof everything involved at a particular situation. ICWE 2016,» *LNCS 9671, Springer,* pp. 476- 483, 2016.

[3] A. Pérez-Vereda, C. Canal, «A People-Oriented Paradigm for Smart Cities,» *International Conference on Web Engineering,* 2017.

[4] D.Luckham, Event Processing for Business: Organizing the Real-Time Enterprise., Hoboken: John Wiley & Sons, 2011.

[5] Foundation OpenStreetMap, «OpenStreetMap,» 1 Julio 2004. [En línea]. Available: https://www.openstreetmap.org/.

[6] EsperTech Inc, «Complex Event Processing, Streaming Analytics, Streaming SQL - EsperTech,» 2006. [En línea]. Available: http://www.espertech.com/.

[7] Apache Software Foundation, «Apache Kafka,» 2011. [En línea]. Available: https://kafka.apache.org/.

[8] Apache Software Foundation, «Apache ZooKeeper,» 2002. [En línea]. Available: http://zookeeper.apache.org/.

[9] Open Data Commons, «Open Database License (ODbL) v1.0,» 25 junio 2009. [En línea]. Available: https://opendatacommons.org/licenses/odbl/1.0/.