UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Un framework para la interacción geolocalizada con el mundo físico a través de dispositivos móviles

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Moriche Montejano, David

Tutor/a: Fons Cors, Joan Josep

CURSO ACADÉMICO: 2022/2023

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Framework para la interacción geolocalizada con el mundo físico a través de dispositivos móviles

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: David Moriche Montejano

Tutor: Joan Josep Fons Cors

2022-2023

Agradecimientos

Resumir cuatro años de carrera en una página es todo un logro, ya que si contáramos todo lo que hemos vivido daría para otro trabajo de muchas páginas. Es por esto que aquí me quiero acordar de las personas que me han acompañado en mayor o menor medida por todo el proceso.

Primeramente, agradecer a mi tutor Joan, que siempre estuvo para ayudarme cuando lo necesitaba y nunca dejó de confiar en mí, ni cuando yo mismo lo dudaba. Me quedo con una frase suya que compartimos por la aplicación Teams, '*En el mundo no es todo blanco y negro, y en los grises hay también esperanza*'.

Seguidamente a mis padres, Isabel y Florencio, que siempre me apoyaron y me dejaron libertad para aprender y formarme de la manera que yo quisiera. Sin su apoyo para conseguir ciertas metas en mi vida, a día de hoy no tendría nada.

También tengo que dar gracias por tener a mi lado a una compañera que me ha ayudado en todo el camino, quedándose conmigo los días y las tardes a mi lado ayudándome a finalizar este trabajo. Dasha, gracias por todo lo que haces por mí.

Por último, pero no menos importante, a mis compañeros de carrera, cuyo apoyo ha sido muy necesario los días que no entendía las cosas y me las explicaban o viceversa. David, Alex, Rubén, Ricardo, os deseo el mejor de los futuros como profesionales del mismo sector.

Resumen

Cuando hablamos de interacciones geoespaciales, el primer pensamiento que se nos puede venir a la cabeza son los dispositivos inteligentes que tenemos en nuestros hogares, también llamados dispositivos IOT. Pues para estos dispositivos las interacciones geoespaciales se están volviendo día tras día en algo más y más común. Desde encender la calefacción cuando se ha detectado que el usuario ha entrado en un área delimitada por él mismo, como podría ser en un radio de 500 metros de su casa, hasta la posibilidad de enviar una notificación al usuario que, una vez se haya alejado una determinada distancia, se le notifique con el estado de los electrodomésticos que se encuentran encendidos en su hogar, así como la posibilidad de apagarlos. Incluso se podría plantear el uso de la solución propuesta para otros escenarios totalmente distintos. Una alternativa podría ser la salida de un usuario de un establecimiento, al cual se le notificaría con un email del producto que ha comprado, en caso de que lo haya hecho. Si por el contrario no ha comprado ninguno, se le podría impactar con una notificación en la que se le informa de que dispone de cierto descuento en dicha tienda. El objetivo de este proyecto es dar una alternativa al desarrollador de una posible implementación final a las soluciones que se encuentran en la actualidad para distintas interacciones se le pudieran ocurrir. Para ello, se han desarrollado una serie de clases, relaciones y métodos agrupados en un framework siguiendo la estructura de las reglas ECA, los cuáles se deberían emplear como una base para un desarrollo personalizado de distintos escenarios como los anteriormente mencionados o escenarios futuros que no se han planteado en primera instancia pero que posteriormente se podrían desarrollar utilizando las mismas clases y métodos. Además, se han propuesto dos posibles escenarios reales para los que se ha implementado una solución utilizando los elementos del framework desarrollado. Con todo esto descrito, finalmente se han expuesto una serie de conclusiones a las cuáles se ha llegado con este proyecto.

Palabras clave: interacciones geoespaciales, framework, IOT, posición, reglas ECA.

Abstract

When we talk about geospatial interactions, the first thought that comes to mind are the smart devices we have in our homes, also called IOT devices. For these devices, geospatial interactions are becoming increasingly common day after day. From turning on the heating when it has been detected that the user has entered an area delimited by himself, as it could be in a radius of 500 meters from his home, to the possibility of sending a notification to the user that, once a certain distance has been removed, you are notified of the status of appliances that turn on in your home, as well as the possibility to turn them off. One might even consider using the proposed solution for completely different scenarios. An alternative could be the departure of a user from an establishment, to which they would be notified with an email of the product they have purchased, if they have done so. If you haven't purchased any, you may be affected by a notification that you have a certain discount at that store. The objective of this project is to give the developer an alternative to a possible final implementation of the solutions currently available for different interactions. To this end, a series of classes, relationships and methods have been developed, grouped into a framework that follows the structure of the ECT standards, which should be used as a basis for a personalised development of different scenarios such as those mentioned above or future scenarios which have not been raised in the first instance but which could be developed later using the same classes and methods. In addition, two potential real scenarios have been proposed for which a solution has been implemented using elements of the framework developed. With all this described, finally a series of conclusions have been exposed to which has been reached with this project.

Keywords : geospatial interactions, framework, IOT, position, ECA rules.

Tabla de contenidos

Índice

1. Introducción

En este proyecto se pretenden desarrollar una serie de clases y métodos para facilitar al desarrollador la creación de diferentes interacciones geoespaciales, así como la acción que se pretenda hacer cuando cierta interacción se lleve a cabo.

Para ello, se ha planteado utilizar el lenguaje Java como lenguaje de programación único en todo el desarrollo, por ser este el más utilizado por el alumno durante todo el curso académico y ser con el que más cómodo y familiarizado se siente él.

Por otra parte, se ha utilizado el protocolo de mensajería MQTT para la recepción de la información del dispositivo móvil con el cual se han hecho las pruebas, por ser un protocolo sencillo de utilizar y de fácil comprensión. Este protocolo se utiliza mediante la aplicación Owntracks, la que nos permite enviar con el patrón publicador/suscriptor mensajes[4][5] con la posición del usuario.

Finalmente, este proyecto se ha basado en el modelo evento-condición-acción (modelo ECA) para desarrollar los diferentes componentes que constituyen la solución final.

1.1 Motivación.

Después de todo lo descrito anteriormente, la motivación principal de este proyecto es que el alumno se familiarice tanto con el desarrollo de un framework con el lenguaje de programación Java, con el protocolo de mensajería MQTT y con las interacciones geoespaciales. A su vez, también se pretende abordar el desarrollo de un proyecto de principio a fin, con el planteamiento de ideas, el desarrollo y sus conclusiones.

Cabe destacar también las motivaciones ecológicas, ya que siendo estos últimos años de suma importancia para el cambio climático que se plantea en un futuro no muy lejano, se entiende que este proyecto podría ayudar a esta causa y tener un gran impacto en el ahorro de energía y agua de los diferentes hogares, o el ahorro de emisión de gases invernadero emitidos por los vehículos particulares. Todo ello se mostrará en los escenarios propuestos al final de este mismo trabajo.

1.2 Objetivos.

El objetivo final al cual este trabajo pretende llegar es tener un framework capaz de interactuar con la posición del usuario y diferentes puntos de interés definidos por el desarrollador, y a su vez ser muy versátil ya que pretende dar total libertad al programador con las acciones llevadas a cabo cuando una cierta acción se ha llevado a cabo. Para alcanzar dicha meta, se han planteado los siguientes objetivos:

1. Crear el código para hacer posible la recepción de los envíos de los mensajes MQTT que son enviados por la aplicación Owntracks con la posición del usuario.
2. Llevar a cabo un estudio de las diferentes interacciones geoespaciales, siguiendo el modelo ECA mencionado anteriormente, que puedan ocurrir en el mundo real, así como plantear la condición que se debe hacer para que dicha interacción suceda.
3. Desarrollar el código necesario para recoger la latitud y la longitud del usuario que la aplicación Owntracks envía en cada uno de sus mensajes.
4. Implementar las soluciones estudiadas anteriormente para hacer posible la evaluación de manera correcta de las distintas interacciones planteadas.
5. Desarrollar una interfaz que sirva de base para la implantación de cualquier tipo de acción que se planteé realizar una vez se ha evaluado una condición como cierta.
6. No depender de ningún programa o tecnología externa para desarrollar el framework.

1.3 Metodología.

La metodología que se ha empleado durante toda la vida del proyecto ha consistido en definir las fases necesarias para crear adecuadamente la solución final. Una vez planteadas las fases, se ha decidido avanzar alcanzando pequeños hitos semanalmente hasta dar por finalizado el proyecto. Se han tenido charlas con el tutor para ver avances, posibles atascos, dudas, explicaciones, etc.

1.3.1 Fase de análisis

En esta primera fase de análisis, se han estudiado diferentes entornos de programación para ver las ventajas e inconvenientes de cada uno y de esta forma elegir el que mejor se adapte a la solución planteada. A su vez, se ha estudiado una serie acotada de interacciones que podría llegar a realizar el usuario con su entorno. Se ha evaluado cuales podrían ser más interesante para

su desarrollo y han sido plantadas siguiendo el modelo ECA mencionado anteriormente.

1.3.2 Fase de desarrollo

Una vez finalizada la fase de análisis de los diferentes entornos y las interacciones, en la cual se habrán elegido las mejores opciones para implantar en la solución propuesta, se pasará a la fase de desarrollo. El objetivo principal de esta fase es dar una posible implantación de todas las clases y métodos necesarios para el framework. Esta fase es la más importante de todo el proyecto, pues es en la que se van a definir los componentes necesarios para el framework, así como la relación entre ellos. Así pues, se han creado las clases que se muestran en la siguiente imagen.

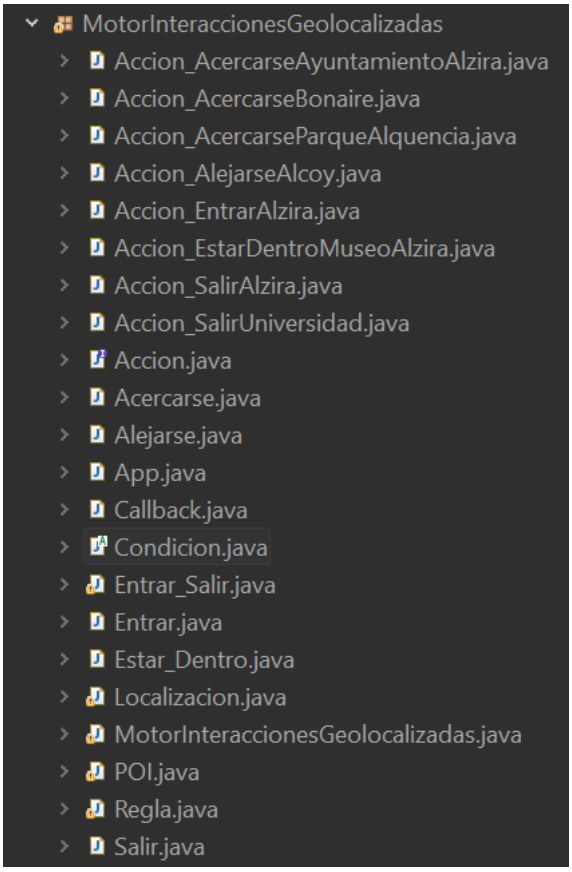

Figura 1. Clases del Proyecto

Como se aprecia en la figura 1, se han creado varias clases para poder llevar a cabo este proyecto. Más adelante se hará una descripción de cada una de ellas y se abordarán detalladamente aquellas que resulten de alto interés para la correcta comprensión del trabajo.

2. Contexto Tecnológico.

Desarrollar un framework relacionado con las interacciones geoespaciales no es una labor sencilla. Actualmente se pueden encontrar soluciones semejantes a la que se plantea en este proyecto, pero son soluciones de pago ligadas a una empresa privada. Es por esto por lo que, como se ha comentado anteriormente, se debe tener en cuenta que herramientas son las adecuadas para llevar a cabo su desarrollo. En esta elección se ha tenido en cuenta varios aspectos que se han considerado de gran importancia.

El primero de ellos es la familiaridad que tiene el estudiante con cada herramienta, lo que se traduce en una mayor agilidad a la hora de desarrollar el proyecto y un mejor manejo de los posibles errores que puedan ocurrir.

El segundo aspecto para tener en cuenta es la documentación de cada una de las herramientas, puesto que una buena documentación es de gran importancia para poder entender las limitaciones que tiene cada una de ellas.

Finalmente, el último aspecto a considerar una vez se han estudiado las diferentes herramientas disponibles en el mercado son las ventajas e inconvenientes que nos brindan cada una de ellas.

Una vez se ha tenido en cuenta todo lo mencionado anteriormente, se han escogido las siguientes herramientas para el desarrollo de este proyecto.

2.1 Entorno de Desarrollo.

Como entorno de desarrollo se ha escogido Eclipse de entre todos los que se han barajado como posibles IDEs de desarrollo. Eclipse proporciona un entorno de desarrollo intuitivo y nos aporta una serie de facilidades a la hora de ejecutar una aplicación como la nuestra.

2.2 Aplicación Owntracks.

Owntracks es una aplicación móvil gratuita disponible tanto en iOS como en Android. Esta aplicación es capaz de gestionar de diversas formas el envío de la localización del dispositivo. Se deben destacar dos características importantes que se han tenido en cuenta a la hora de escoger esta aplicación para hacer el envío de la localización del usuario.

Primeramente, se ha escogido esta aplicación ya que funciona con el paradigma publicador/subscriptor[4], el cuál es conocido por el alumno y permite usar colas de mensajería para enviar mensajes[5] sin necesidad de tener una

conexión punto a puno, ya que la aplicación envía la localización al broker MQTT.

La segunda característica que se ha tenido en cuenta y que viene muy relacionada con lo anteriormente comentado, la aplicación utiliza el protocolo de mensajería MQTT, un protocolo ligero de envío de mensajes. Es un protocolo muy estandarizado a lo largo de todos los dispositivos IoT[1].

2.3 Librería Paho.

La librería Paho es una de las más utilizadas a la hora de desarrollar proyectos basados en MQTT[2]. Es una librería de código abierto creada por la fundación Eclipse para desarrollar proyectos en Java y que puedan ser utilizados con la JVM u otra plataforma compatible con Java como podría ser Android[3].

Se ha decidido usar esta librería para hacer la conexión con el servidor [13] y así poder recibir el mensaje que nos llega desde el dispositivo del usuario.

A continuación, se pretende mostrar con unas simples líneas de código como se ha realizado dicha conexión utilizando las clases y métodos propios de la librería Paho. Hay que remarcar que se ha decidido eliminar los archivos de persistencia que se generan por cada conexión que realiza el protocolo para ahorrar espacio en el disco.

```
public void setOwntracks(String url, String topic) {
try {
    MqttDefaultFilePersistence persistence = null;
    try {
        persistence = new MqttDefaultFilePersistence("/tmp");
    }catch (Exception e) {}
    if ( persistence != null ) {
        myClient = new MqttAsyncClient(url, UUID.randomUUID().toString(), persistence);
    }else {
        myClient = new MqttAsyncClient(url, UUID.randomUUID().toString());
    }
    Callback callback = new Callback();
    myClient.setCallback(callback);
    IMqttToken token = myClient.connect();
    token.waitForCompletion();

    myClient.subscribe(topic, 0);
 } catch (MqttException e) {
     System.out.println(e.toString());
     }

}
```

Figura 2: Configuración de la conexión con el servidor.

```
MotorInteraccionesGeolocalizadas motor = new MotorInteraccionesGeolocalizadas();
motor.setOwntracks("tcp://tambori.dsic.upv.es:10088", "owntracks/#");
```

Figura 3: Configuración de la conexión con el servidor.

[1] https://aprenderbigdata.com/mqtt-mosquitto/
[2] https://sacavix.com/2020/02/mqtt-publish-subscribe-en-tiempo-real-ejemplo-java/
[3] https://www.eclipse.org/paho/index.php?page=clients/java/index.php

2.4 Java.

Finalmente, se ha decidido utilizar el lenguaje de programación Java frente a el resto de las posibilidades que encontramos actualmente en el mercado. Esto se debe al alto conocimiento que tiene el autor de este proyecto con el lenguaje, pues es el que más ha utilizado a lo largo de toda su vida estudiantil. Java es uno de los principales lenguajes de programación orientada a objetos[4]. Desde su lanzamiento en 1995, se ha convertido en uno de los lenguajes más importante en el mundo de la programación, habiéndose construido sobre él una gran cantidad soluciones.

[4] https://universidadeuropea.com/blog/programacion-orientada-objetos/

3. Análisis del problema

Para analizar bien la problemática y entender el problema al que se pretende dar solución, deberíamos entender primeramente que son las interacciones geolocalizadas.

En ocasiones, nuestras actividades cuotidianas implican el hecho de que debamos cambiar nuestra posición a un cierto lugar y tiempo [9]. Dicha información puede llegar a ser de gran utilidad, pues las acciones que se podrían llegar a plantear con la información acerca de nuestra posición resultarían de gran ayuda para automatizar ciertos aspectos de nuestro día a día [11]. De esta forma resultaría interesante, por ejemplo, el hecho de recopilar la información acerca de cuándo un usuario abandona su puesto de trabajo en la oficina para dirigirse a casa, con el fin de ejecutar una cierta acción de su propio interés. Por otra parte, en el caso de que un usuario se acerque o aleje de una posición específica. Además, no existe la necesidad de limitar dichas acciones al hecho de ser individuales, de esta forma otra acción que solemos hacer es reunirnos con nuestros familiares o amigos, de lo cual también se podría extraer una valiosa información para ejecutar una cierta acción.

Este trabajo pretende recopilar este tipo de acciones que se realizan a menudo y abstraerlas, definirlas en un lenguaje que nos permita identificar y procesar que es lo que está sucediendo exactamente con la localización del usuario. En apartados posteriores, se le dará un contexto tecnológico a este lenguaje que se pretende definir a continuación, así como llegar a implementar alguna de las sus interacciones.

3.1 Interacciones

3.1.1 Interacción 1: Estar dentro del área de un objeto

Esta sería una de las interacciones más sencillas de entender. Lo primero que debería suceder es el envío del mensaje en el que se incluye la posición por parte del objeto2 a una cierta localización. Posteriormente, cierto proceso debería recoger los datos del mensaje y procesarlos. Tras evaluar los datos que se han recibido y compararlos con los datos que ya se tienen de un cierto objeto1, se comprueba que la localización del usuario se encuentra dentro de la posición de un cierto objeto posicionable.

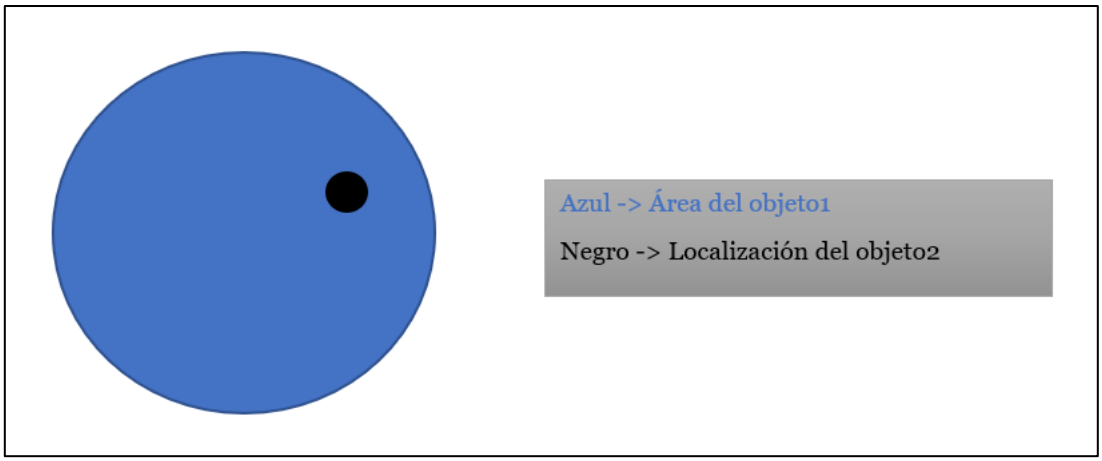

Figura 4: Esquema gráfico interacción 1

3.1.2 Interacción 2: Estar fuera del área de un objeto

Esta sería otra de las interacciones más sencillas de entender. El evento que desencadena todo es el mismo que el resto de las interacciones, el envío de la posición por parte del objeto2. El proceso de análisis de la información recibida, tras haber analizado los datos que tiene acerca de los dos objetos, evalúa la condición de esta interacción como verdadera. Esto se debe a que la posición del objeto2 que viene definida de una cierta manera no se encuentra dentro del área del objeto1.

Figura 5: Esquema gráfico interacción 2

3.1.3 Interacción 3: Entrar al área de un objeto

Esta interacción y las siguientes son las más complejas, ya que a partir de aquí se deberá involucrar la posición anterior del objeto2 y la nueva posición. Esto se traduce en que se ha de guardar la posición recibida por el mensaje anterior y la posición recibida en el nuevo mensaje. Una vez se tienen ambas localizaciones, se deberá añadir dicha información al análisis de la condición. En este caso, si la

posición anterior del objeto2 respecto al área del objeto1 está fuera, pero la nueva posición está dentro, se cumple la condición y se evalúa como verdadera.

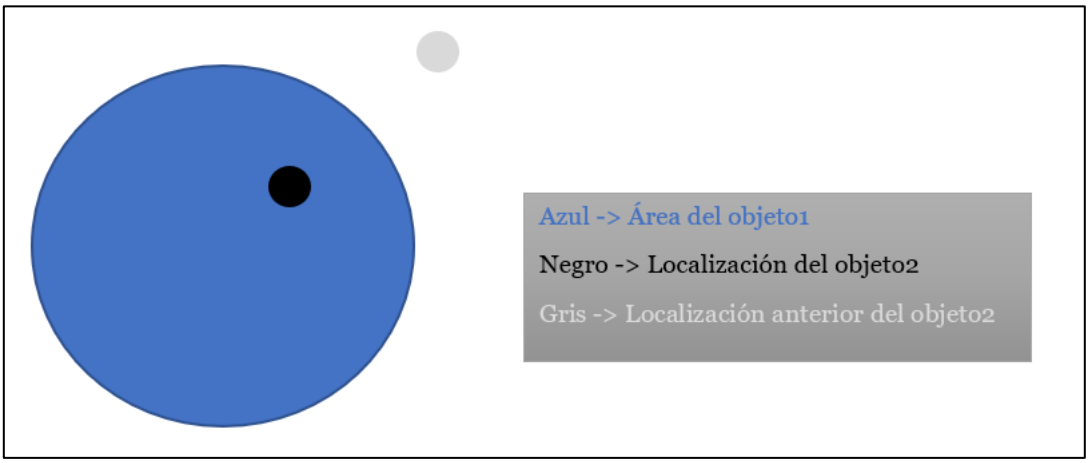

Figura 6: Esquema gráfico interacción 3

3.1.4 Interacción 4: Salir del área de un objeto

En esta interacción, como en la anterior, se deberá hacer el mismo proceso de guardado de las posiciones. Estos datos deberán añadirse a la comprobación de la misma forma que en la interacción anterior. En este caso, si la posición anterior del objeto2 respecto al área del objeto1 está dentro, pero la nueva posición está fuera, se cumple la condición y se evalúa como verdadera.

Figura 7: Esquema gráfico interacción 4

3.1.5 Interacción 5: Acercarse al área de un objeto

En las sucesivas explicaciones se dará por sentado que se deberá guardar las posiciones e involucrarlas en la condición. En este caso, si la posición anterior del objeto2 respecto al área del objeto1 está más lejos que la nueva localización, se cumple la condición y se evalúa como verdadera.

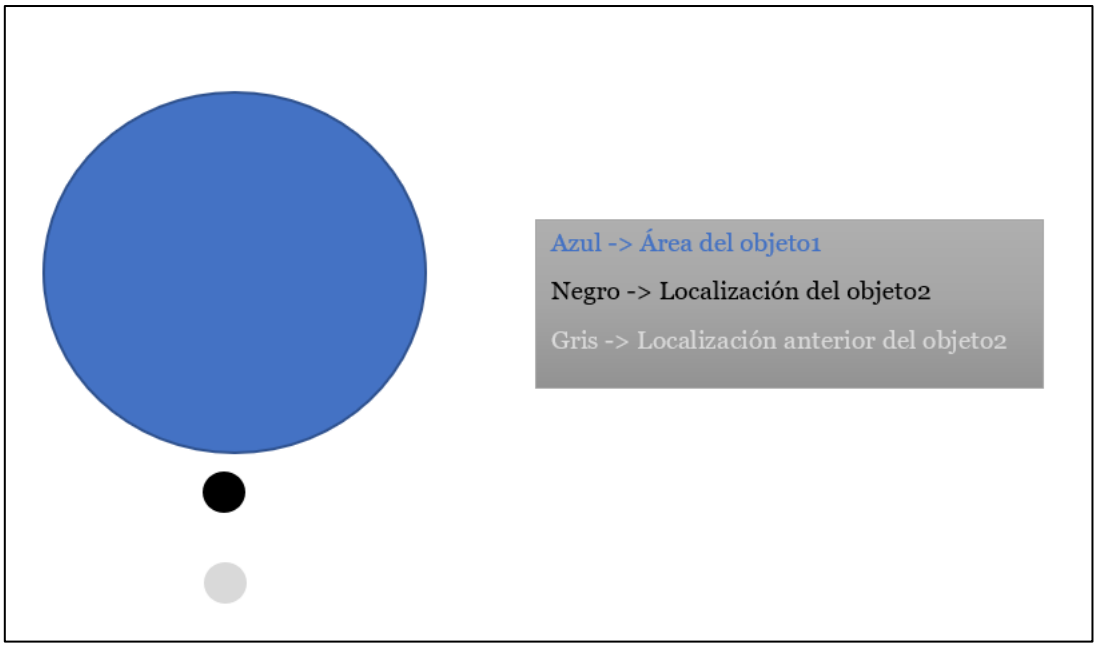

Figura 8: Esquema gráfico interacción 5

3.1.6 Interacción 6: Alejarse del área de un objeto

La condición de esta interacción se evaluará como verdadera cuando la posición anterior del objeto2 se encuentre más cerca del área del objeto1 que la nueva posición recibida.

Figura 9: Esquema gráfico interacción 6

3.1.7 Interacción 7: Quedarse estático estando fuera del área de un objeto

En este caso, al superponerse ambas posiciones, se ha decidido mostrar la nueva localización respecto a la antigua. Esta es la razón de que no exista un punto gris en la imagen anterior. Esta interacción sucederá y su condición se evaluará como verdadera cuando la posición anterior del objeto2 sea la misma que la nueva posición recibida y ambas estén fuera del área del objeto1.

Figura 10: Esquema gráfico interacción 7

3.1.8 Interacción 8: Quedarse estático estando dentro del área de un objeto

La razón por la que no se visualiza un punto gris es la misma que en la interacción anterior. Así pues, la condición se evaluará como verdadera cuando la posición anterior del objeto2 sea igual que la posición actual del objeto y ambas se encuentren dentro del área del objeto1.

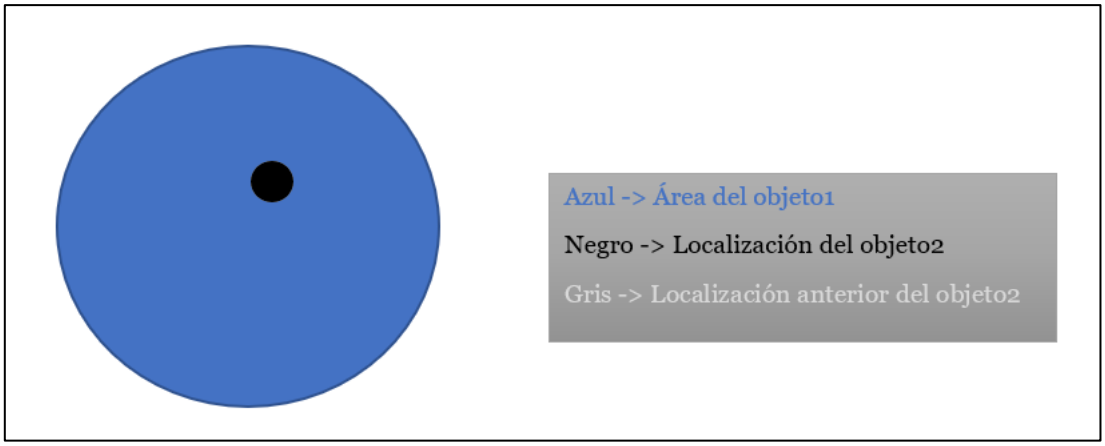

Figura 11: Esquema gráfico interacción 8

Para concluir con este punto, se van a exponer una serie de interacciones diferentes a las anteriores. En todas las interacciones anteriormente mencionadas únicamente se ha tenido en cuenta la posición de un objeto, pero pudiera ser que se tuviera un conjunto de objetos, de los que se tiene la posición, y que interactuasen con otro objeto o con otro conjunto de objetos, no necesariamente con un único objeto. Por ello, se puede extrapolar todas las interacciones anteriores a interacciones con conjuntos de objetos. De esta forma, se tiene que un objeto puede estar acercándose a la posición de otro objeto. Además, se podría dar el caso de que dos objetos se encuentren dentro del área de otro objeto.

Como interacciones con conjuntos de objetos, se plantean las siguientes, teniendo en cuenta que cualquiera de las anteriores también se podrían considerar de este tipo con una serie de modificaciones.

3.1.9 Interacción 9: Un objeto está dentro del área de otro objeto y n están fuera

En este caso, se deberá tener en cuenta lo anteriormente mencionado, se han de tener las posiciones de cada objeto para poder evaluar esta condición. Clarificar que en la imagen se muestran 4 puntos que no están dentro del área, pero podrían ser más puntos o menos. De este modo, si las posiciones del resto de objetos se encuentran fuera del área de otro objeto y la de un objeto está dentro, la condición se evaluará como verdadera.

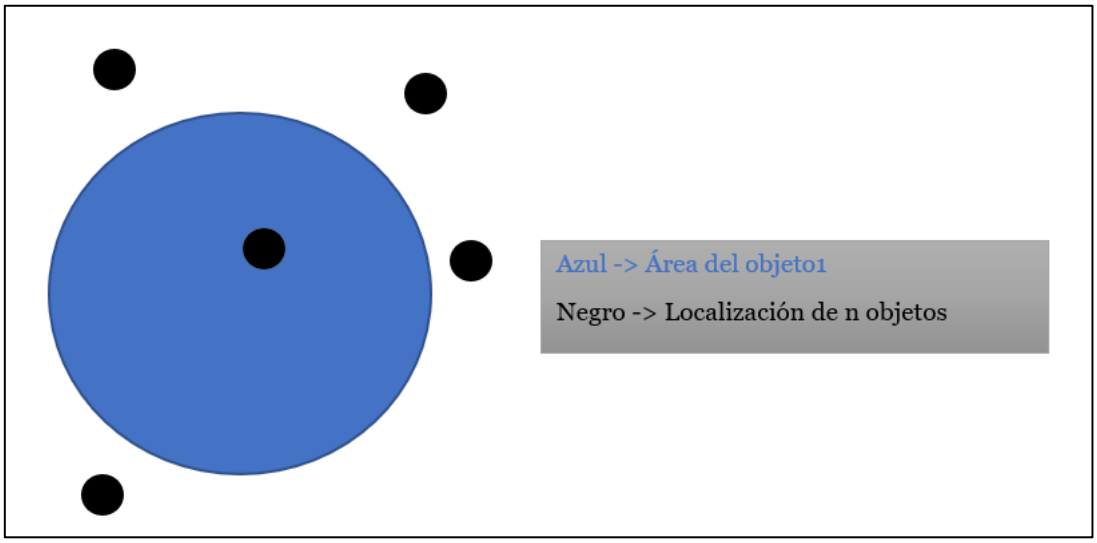

Figura 12: Esquema gráfico interacción 9

3.1.10 Interacción 10: N objetos están dentro del área de otro objeto y uno está fuera

Hay que mencionar que se deberá tener en cuenta lo mencionado en el caso anterior respecto a las posiciones y números de usuarios. Por consiguiente, la condición se evaluará a verdadera cuando todas las posiciones de los objetos se encuentren dentro del área del objeto1 excepto una que se hallaría fuera de dicha área.

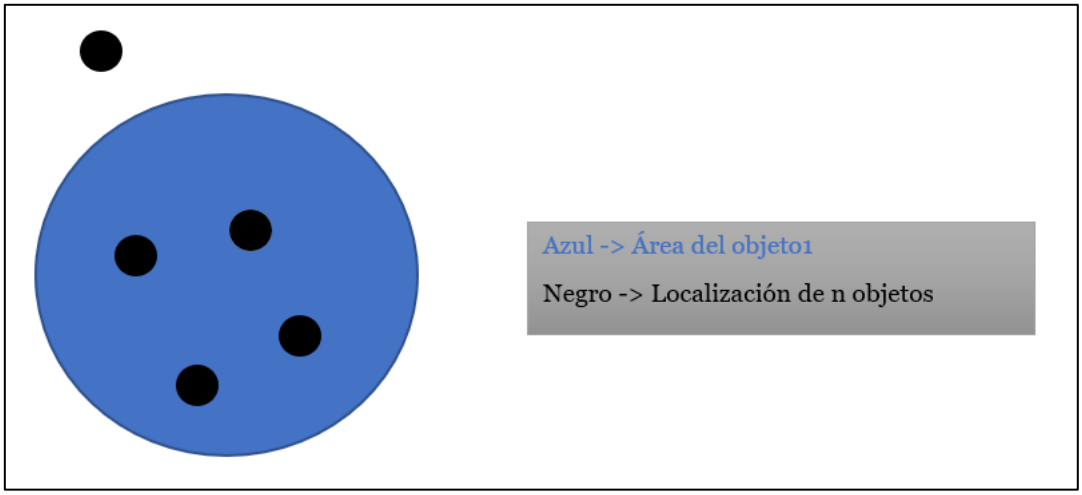

Figura 13: Esquema gráfico interacción 10

3.1.11 Interacción 11: Un objeto se acerca a otro objeto

Las consideraciones previas también son necesarias para este caso. Por tanto, esta interacción tendrá una evaluación verdadera cuando la posición actual del objeto2 se encuentre más cerca del objeto1 que la posición anterior.

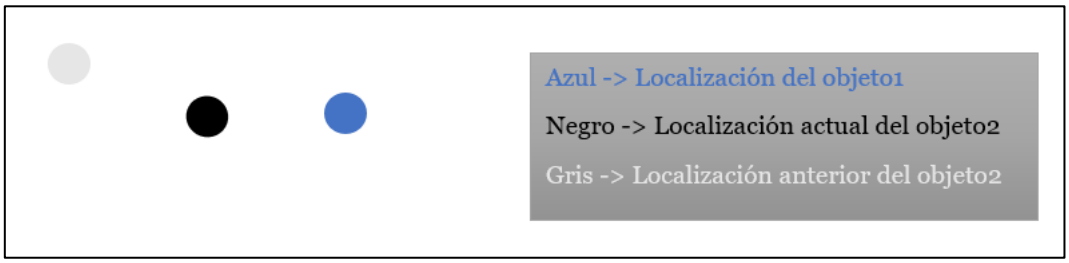

Figura 14: Esquema gráfico interacción 11

3.1.12 Interacción 12: Un objeto se aleja de otro objeto

Hay que mencionar que se deberá tener en cuenta lo mencionado en el caso anterior respecto a las posiciones y números de usuarios. Por tanto, esta interacción tendrá una evaluación verdadera cuando la posición actual del objeto2 se encuentre más lejos del objeto1 que la posición anterior.

Figura 15: Esquema gráfico interacción 12

3.2 Análisis de los elementos

Finalmente, una vez se han expuesto una serie de interacciones que pueden llegar a suceder de manera diaria, vamos a plantear el escenario final de análisis. A continuación, se van a detallar los elementos que se han recogido una vez hecho el estudio del escenario a implementar.

3.2.1 Elementos posicionables estáticos

Este tipo de elemento, como indica su nombre es estático, por lo que su posición no variará a lo largo del tiempo y se deberán declarar una única vez en el código para el que se implemente la solución.

3.2.2 Elementos posicionables dinámicos

El segundo tipo de elemento es el que debe enviar la posición cada cierto tiempo. Estos elementos son los que permitirá hacer un seguimiento de la localización exacta en cada momento del usuario y de esta forma poder aplicar las reglas expuestas en el apartado anterior.

3.2.3 Conjunto de elementos posicionables

Este elemento haría referencia a un conjunto de elementos de los dos tipos anteriores. Podría ser un conjunto único de elementos posicionables estáticos, uno de elementos posicionables estáticos o una mezcla entre estos dos tipos.

3.2.4 Posición

Uno de los elementos más importantes sería la posición de los objetos. Este elemento es el que nos permitirá realizar operaciones sobre localizaciones (acercarse, alejarse, estar en, ...). Por simplicidad, hemos optado por utilizar posiciones GPS (incluyendo latitud y longitud), pero el proyecto podría extenderse con otro tipo de ubicaciones.

3.2.5 Interacción geolocalizada

Las interacciones geolocalizadas son el elemento principal de este proyecto. Estas estarán compuestas por dos elementos principales, uno o varios objetos posicionables y una regla.

3.2.6 Conjunto de interacciones

Este elemento agrupa cada una de las interacciones que se han descrito en el apartado anterior y las almacena. Esto sirve para comparar el patrón de interacción con el que se ha establecido la regla con la localización que se ha recibido y de esta forma saber cuando una interacción ha tenido lugar.

3.2.7 Regla

El elemento regla es el que relaciona cada una de las interacciones con una acción concreta. Esta relación se ha hecho 1-1, pues se ha decidido que quedaría más claro de esta forma. En el futuro desarrollo se ha decidido ampliar esta relación de 1..*, así se podrían llegar a plantear interacciones que se deban cumplir con más de una condición. Para esta versión no se ha considerado ninguna interacción de este tipo.

3.2.8 Acción

Las acciones son eventos o hechos que deben suceder una vez se ha detectado que una interacción ha tenido lugar. En la solución no se ha querido limitar este elemento con una simple acción predeterminada, sino que se ha querido dar la libertad al futuro desarrollador de decidir que acción es la que se quiere realizar en un determinado escenario.

Con todo esto descrito, se pretende hacer un resumen e ilustrar todo lo anterior con la siguiente figura (figura 16), en la cual se exponen como un diagrama de clases lo comentado anteriormente.

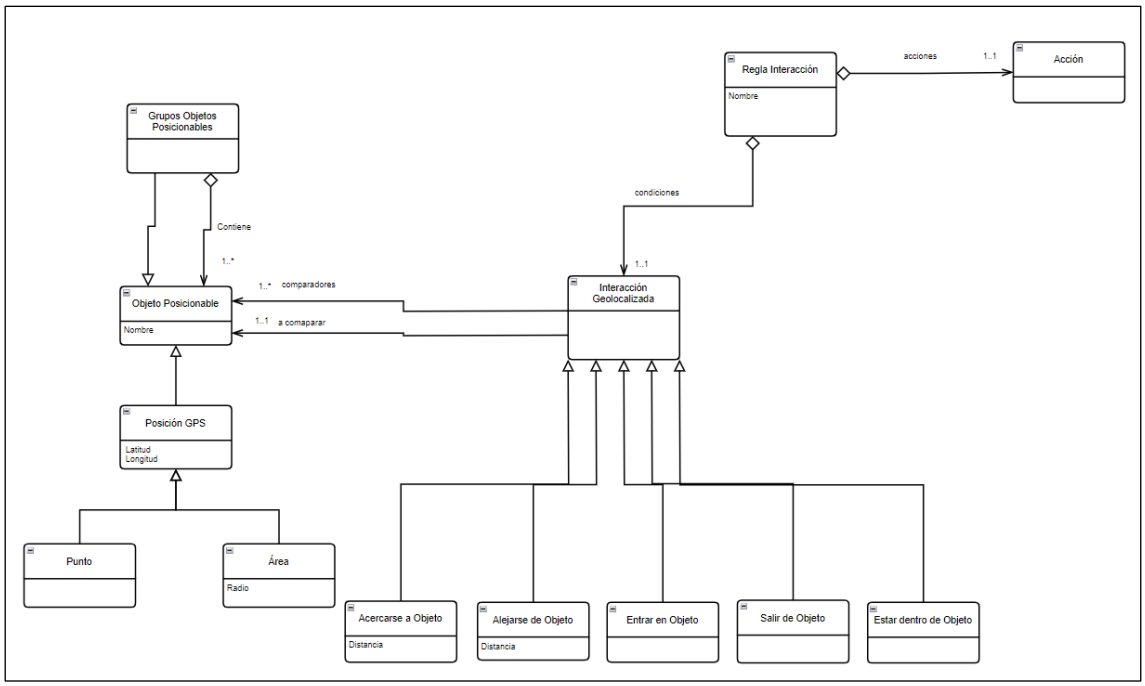

Figura 16: Clases de dominio

En la figura anterior[1][2][3] se aprecian los diferentes conceptos que se han analizado en la problemática inicial, así como las distintas relaciones que guardan unos con otros. Así pues, las interacciones geolocalizadas son el centro de nuestro análisis. Dichas interacciones se componen de reglas para poder llevarse a cabo, con sus respectivas acciones que se deben ejecutar si cierta regla se cumple. Para todo ello, se requieren dos objetos posicionables, uno hará la función de ser el comparador y el otro el comparado. Analizando los objetos posicionables, se ha decidido ubicarlos mediante sus coordenadas GPS, lo que implica que se necesiten la latitud y longitud de los objetos. Además, se han diferenciado entre punto y área, pues los objetos posicionables de tipo punto no tendrán un radio, por lo que su posición será la definida por su latitud y su longitud. Sin embargo, la posición de los objetos posicionables de tipo área se verá representada por la latitud y longitud de dicho objeto más menos su radio.

4. Diseño de la solución

Una vez analizado la problemática inicial, los diferentes componentes que son necesarios para dar soporte a la solución y los requerimientos y limitaciones con las que nos vamos a encontrar a la hora de desarrollar el proyecto, se presentará el diseño planteado. En esta fase se propondrá un diseño que se ha considerado el más acorde con la figura (figura 16) del análisis de la problemática. Para ello, se va a explicar de manera arquitectónica cuál debería ser la forma que debe tomar la solución que se plantea.

A continuación, se mostrarán los distintos componentes por los cuales está formada la solución y como están relacionados entre ellos. Estos componentes guardan una estrecha relación con la figura final (figura 16) del apartado de análisis, pues algunos de los elementos que se han descrito en dicha figura están muy visibles en esta, otros se tendrá que profundizar más para lograr observarlos. Seguidamente, se pretende dar una explicación detallada de cada componente, la funcionalidad que aporta a la solución y posteriormente se explicará cómo sería un flujo de trabajo real.

4.1 Arquitectura de la solución

Las diferentes partes que componen esta solución son las que siguen:

1- **Dispositivo móvil**: Nos serviremos de los dispositivos móviles como una potente fuente de información sobre la geolocalización del usuario[10]. Este componente estaría representado en la figura 16 como un objeto posicionable de tipo punto. Consta de varias funciones de alta importancia. La primera de ellas es clara, enviar la localización del usuario. El mensaje se enviará a un cierto canal de reporte de la posición del usuario para ser recogido posteriormente por el servidor. La segunda función es mostrar el mensaje que el servidor ha devuelto una vez se ha analizado el mensaje enviado previamente por el mismo dispositivo. Como no se ha planteado limitar las acciones que puede llevar a cabo el desarrollador con la solución final, este mensaje podría tener cualquier tipo de estructura o formato. Se deberá escoger el que mejor convenga para la solución final implementada.

2- **Módulo de envío y recepción de mensajes** [5]: Las dos funciones que debe tener este elemento son de suma importancia. Por una parte, este módulo deberá recoger la información que ha enviado el objeto posicionable anteriormente descrito. Seguidamente, este elemento deberá cumplir con su segunda función una vez recibido el mensaje. El

elemento deberá enviar la información recibida al servidor y de esta forma cumplir su segunda función.

Una de las premisas que se deberá tener es que este modelo debe ser escalable, por lo que varios dispositivos deberán poder publicar su información de manera asíncrona con este elemento de la solución.

3- **Módulo de procesamiento**: Este componente es el más interesante para la solución propuesta, pues será él quien contenga el framework desarrollado para evaluar los diferentes eventos que envía el dispositivo móvil. Este componente recibe los datos publicados por el objeto posicionable y los guarda en una variable que se adapta a un lenguaje entendible para su procesamiento. Dicho procesamiento se ha visto modelado gracias a las reglas ECA. Estas reglas sugieren que, en el momento en el que se recibe un evento, en nuestro caso sería la recepción del mensaje enviado, se deben procesar los datos para ser evaluados por una serie de condiciones previamente establecidas por el desarrollador de la solución.

   En el caso propuesto, estas condiciones son todas aquellas interacciones geoespaciales que se nos puedan llegar a ocurrir: estar dentro, estar fuera, alejarse, acercarse, salir, entrar, estar cerca, estar lejos, moverse estando dentro de una cierta localización, etc. Como se aprecia, hay numerosas interacciones espaciales que se han tenido en cuenta a la hora de crear esta solución. Más allá de las anteriormente mencionadas, cabe mencionar que se nos pueden ocurrir condiciones complejas, no únicamente se tiene porque limitar a estar dentro o estar fuera de algo. Con la solución propuesta se podrían llegar a plantear interacciones como alejarse y salir de un objeto o acercarse y entrar a un objeto. Las pruebas sí que se han limitado a condiciones simples, pero el código está desarrollado para brindar esa capacidad si es que fuera necesario.

   Finalmente, una vez procesados los datos que se han enviado en el mensaje al servidor, este elemento debe dar una respuesta dependiendo de la evaluación hecha a esos datos. Hay varias situaciones que deberían tenerse en cuenta. La primera de ellas es la posibilidad de que no se cumpla ninguna de las condiciones establecidas en el código del framework. En este caso el futuro desarrollador deberá considerar que se quiere hacer, ya que en la solución propuesta se ha optado por no enviar ningún tipo de mensaje de respuesta. De estar de acuerdo con lo propuesto no es necesaria ninguna modificación, en caso contrario se deberían hacer las modificaciones necesarias para dar soporte a esta casuística.

La segunda posible situación es que se cumpla una de las condiciones. En este caso no habría problema, puesto que al cumplirse una única, no se tiene la necesidad de priorizar o descartar un mensaje a enviar. El servidor ejecutará la acción correspondiente a la condición que se ha evaluado como verdadera. Se debe hacer énfasis en que con la solución propuesta no se ha planteado limitar la acción que se debe ejecutar cuando una cierta condición se haya evaluado como verdadera. Este código se ha dejado como solución abierta para posteriormente desarrollar una acción personalizada a la solución que pretenda hacer la persona que haya decidido utilizar el framework propuesto. En el caso de la problemática que se ha planteado, se deberá publicar un mensaje en la cola de mensajería con las distintas ofertas que se han recogido una vez se ha consultado en la base de datos interna del servidor.

Por último, la última situación posible se da cuando una o más condiciones son evaluadas como verdaderas. En este caso se deberá estudiar qué es lo que se pretende hacer ya que, dependiendo de la solución final que se deseé implementar con el código desarrollado, podría tener problemas si no se desea ejecutar más de una acción por condición evaluada. Si la naturaleza de la aplicación final contempla la posibilidad de ejecutar varias acciones en el caso que más de una condición sea verdadera, no será necesaria ninguna modificación adicional.

Hay que mencionar que existe la posibilidad de aplicar todas las reglas descritas anteriormente en un ámbito más amplio, y se podría decir que más real, en el que no solo sea un dispositivo el cuál envíe los datos al módulo de procesamiento, sino que fuesen varios dispositivos enviando mensajes asíncronamente mediante el uso de cada una de sus colas de mensajería individuales. Si se plantea este nuevo escenario se podrían dar más condiciones que no se han planteado previamente, como la posibilidad de que dos dispositivos estén juntos, que dos dispositivos estén separados, que un dispositivo se esté acercando o alejando a otro dispositivo específico, etc.

Para dar soporte a todo este nuevo conjunto de interacciones geoespaciales que tienen entre dispositivos no haría falta modificar el código del framework. El único requerimiento nuevo que surge es estudiar la necesidad de memoria de procesamiento que se debería utilizar en este elemento para dar soporte a tales requerimientos. En el caso planteado se ha decidido que dicho estudio no recae en el ámbito de este proyecto.

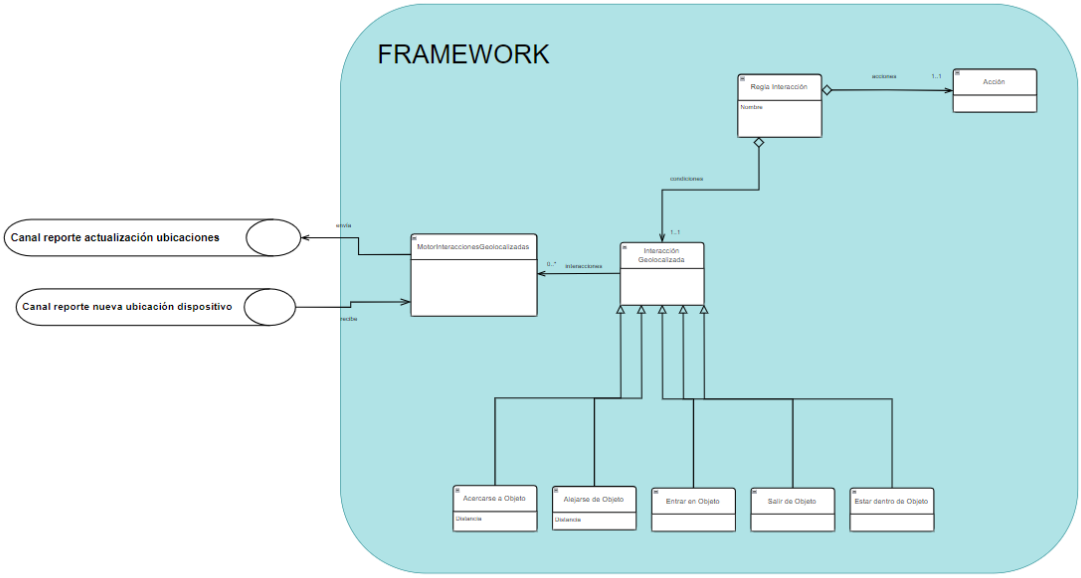

Figura 17: Esquema del framework

4- **Módulo de almacenamiento**: Este elemento será el encargado de almacenar en una memoria interna dos datos de gran importancia para el procesamiento. Por un lado, deberá guardar la localización actual de cada uno de los elementos posicionables. De esta forma, se estará guardando tanto la posición actual que tengan los objetos posicionables con una localización variable, como puede ser un dispositivo móvil, y a su vez se guardará la latitud y longitud de los objetos posicionables cuya posición no sea variable, como puede ser un hospital o un centro comercial. Por otro lado, se deberá guardar la posición anterior de los objetos posicionables con una localización variable, puesto que este dato será necesario para realizar la comprobación de ciertas interacciones, como pueden ser "entrar" y "salir".

5- **Mensajes de posiciones**: Por lo referente a los mensajes que se necesitan intercambiar por parte del módulo de procesamiento y el objeto posicionable, estos deberían incluir un mínimo de dos cosas para poder llevar a cabo la evaluación de la posición. Las dos cosas que se deberán recibir son la latitud y longitud del objeto posicionable, para de esta forma almacenarlas y compararlas con el resto de los objetos posicionables. En caso de que el objeto posicionable fuese de tipo área, también sería necesario incluir a los dos anteriores el radio de dicha área, para de esta forma añadirlo a la evaluación.

Finalmente, se expone una figura (figura 18) a modo ilustrativo del diseño de la solución.

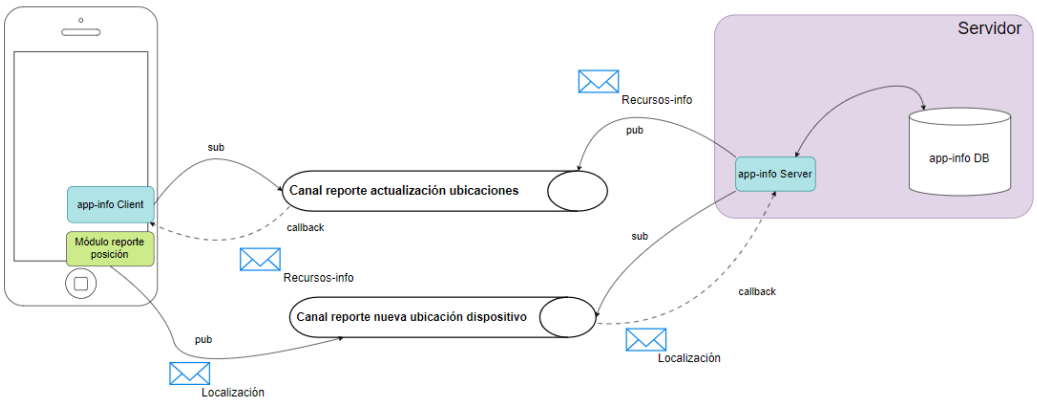

Figura 18: Diagrama de diseño de la aplicación

4.2 Framework para interacciones geolocalizadas

Seguidamente se van a describir las diferentes clases que se han creado, así como las distintas relaciones que tienen unas con otras y alguno de los métodos más interesantes de alguna de ellas.

4.2.1 Motor Interacciones Geolocalizadas

La clase Motor Interacciones Geolocalizadas tiene varias funciones muy importantes. La primera de ellas es establecer la conexión [13] necesaria para recibir y enviar los mensajes al módulo de recepción. La segunda función es inicializar el proceso de evaluación de los datos recibidos. Por último, deberá enviar un mensaje de respuesta acorde a la evaluación realizada. Con todo esto, la clase deberá dar tener los siguientes atributos.

- **Cliente conexión módulo envío/recepción**: Este atributo de la clase será el encargado de aportarnos las funcionalidades necesarias para poder establecer la conexión con el módulo de envío/recepción de mensajes.
- **Reglas**: El atributo “Reglas” será el encargado de almacenar cada una de las condiciones que implantan una interacción junto con cada una de las acciones que se quieran llevar a cabo si la regla que se está evaluando dentro del conjunto de reglas es cierta.
- **Localización**: Este atributo será el encargado de recoger los datos recibidos por parte del módulo de envío/recepción y transformarlos en algo que el propio programa entienda.

Una vez descritos los atributos de la propia clase, se van a exponer los procesos mediante los que se pretende dar soporte a las funcionalidades anteriormente

descritas. Así pues, se necesitará un método de conexión con el módulo de envío/recepción: Este método utilizará las funcionalidades que nos aporta el primero de los atributos descritos para establecer un canal de envío y recepción de mensajes. El siguiente método deberá almacenar todos los objetos posicionables que el desarrollador final haya inicializado y sobre los cuales se quieran llevar a cabo las evaluaciones respecto a otros objetos posicionables. Otro de los métodos que deberá implementar esta clase es uno que almacene en el atributo "Reglas" todas las relaciones entre todas las interacciones y sus respectivas acciones. Finalmente, se necesitará un método que inicialice todo el procesamiento de la información una vez se haya recibido un mensaje.

Con todo esto, la clase MotorInteraccionesGeolocalizadas deberá tener los siguientes atributos y métodos.

Clase	Atributos	Métodos
MotorInteraccionesGeolocalizadas	ClientConexion cliente List<Regla> reglas Localizacion localizacion	setConexion() MotorInteraccionesGeolocalizadas() addObjetoPosicionable(OP) addRegla(Regla) inicializar(Mensaje)

Figura 19: Clase MotorInteraccionesGeolocalizadas

4.2.2 Localizacion

La clase Localización será sobre la cual se guarde la localización del usuario. Esta clase deberá tener una serie de atributos necesarios para dar soporte a las interacciones mencionadas anteriormente. Estos atributos son:

- **Latitud**: Este atributo tiene el valor de la latitud actual del usuario enviada en el mensaje y recogida por el método de la clase anterior.

- **Longitud**: En este atributo se guarda el valor de la longitud actual del usuario que, al igual que el atributo anterior, ha sido recogido gracias al método de la clase anterior.

- **Latitud_Anterior**: La función de este atributo radica en la necesidad de guardar la latitud anterior del usuario para lograr las interacciones en las cuales necesitamos este dato.

- **Longitud_Anterior**: Al igual que el atributo anterior, es necesario guardar la longitud anterior del usuario en una variable para poder dar soporte a las interacciones que requieren este dato.

- **Nombre**: Este atributo se ha decido poner para trabajos futuros, para de esta forma poder diferenciar entre usuarios o dispositivos.

Junto con los atributos anteriores, se deberán añadir los métodos setter y getter de cada atributo y un constructor de la clase. Así pues, el esquema de esta clase es el que sigue.

Clase	Atributos	Métodos
Localizacion	Double Latitud Double Longitud Double Latitud_Anterior Double Longitud_Anterior String Nombre	Localizacion(Latitud,Longitud,LA_ANT,LO_ANT,Nombre) setLongitud(Longitud) getLongitud() setLatitud(Latitud) getLatitud() setLatitud_Anterior(Latitud) getLatitud_Anterior() setLongitud_Anterior(Longitud) getLongitud_Anterior() setNombre(Nombre) getNombre()

Figura 20: Clase Localizacion

4.2.3 POI

POI.java es la clase que comprende una de las labores primordiales en este proyecto. Esta clase representa todas las áreas sobre las cuales se quiere hacer las comprobaciones necesarias para evaluar si una interacción ha tenido lugar o no. Es por esto por lo que los atributos de esta clase son los que siguen.

- **Latitud**: El atributo latitud, como su nombre indica, representa la latitud del POI específico.

- **Longitud**: Así mismo, este es el encargado de almacenar la longitud del POI específico.
- **Nombre**: El atributo nombre contiene un alias que el desarrollador ha querido otorgar al área específica.

- **POIs**: Este atributo es un poco especial. Es el encargado de almacenar cada uno de los POIs que se han declarado en una variable para poder acceder a él a la hora de evaluar el conjunto de reglas previamente establecidas.

- **Radio**: La labor de este atributo consiste en almacenar el radio establecido por el desarrollador para crear el área del propio POI.

Una vez listados los atributos que tiene la clase POI, se van a explicar los métodos que son necesarios implementar en esta clase. Para ello se deberá tener en cuenta los requerimientos que debe cumplir esta clase. Algunos de ellos son

claros, el método constructor y los métodos setter y getter de los atributos anteriores para poder acceder a ellos desde otras clases. También se deberá implementar un método mediante el cual se guarden todos los POIs declarados por el desarrollador en el atributo POIs, y a su vez otro mediante el que se pueda recuperar todos los POIs.

Haciendo una recopilación de todo lo anterior, el esquema de la clase POIs.java es este.

Clase	Atributos	Métodos
POI	Double Latitud Double Longitud Double Radio String Nombre POI[] POIs	POI(Latitud,Longitud,String,Radio) setLongitud(Longitud) getLongitud() setLatitud(Latitud) getLatitud() setLatitud_Anterior(Latitud) getLatitud_Anterior() setLongitud_Anterior(Longitud) getLongitud_Anterior() setNombre(Nombre) getNombre() savePOIs(POI) getPOIs() addPOI(POI)

Figura 21: Clase POI

4.2.4 Regla

Esta clase es la encargada de relacionar las dos clases siguientes, Condición y Acción. Esta relación tiene una relación de aspecto 1:1, es decir, una regla se compone de dos elementos, una condición y una acción. Por consecuente, a la hora de crear una regla, deberemos crear una condición específica para dicha regla, además de una acción que ejecutar cuando la evaluación de la condición sea positiva. Para conseguir lo que se ha descrito anteriormente, los atributos requeridos para esta clase son los siguientes.

- **Condicion**: Este atributo es una instancia de la clase Condicion, representa la condición que se ha de cumplir para que se ejecute la acción correspondiente.

- **Accion**: El segundo atributo es otra instancia de otra clase, pero en este caso de la clase Accion. Este atributo hace referencia a la acción correspondiente que se ha de ejecutar siempre y cuando la condición de la regla se evalúe como verdadera.

- **Descripcion**: El último atributo de esta clase se ha añadido para dar un poco de significado a la propia regla. Este atributo simboliza una descripción breve de la regla.

Como se ha hecho anteriormente, una vez se han listado los distintos atributos que tiene la clase, se han de analizar los requerimientos necesarios a los que esta clase debe dar solución. Así pues, los primeros dos métodos deberían ser los que brinden la forma de añadir una condición y una acción respectivamente. Otra de las funciones para las que esta clase debería dar soporte es la manera de ejecutar la evaluación de las condiciones asociadas a una regla específica. Finalmente, si la condición de la regla específica se evalúa como verdadera, se deberá implementar un método mediante el cual se envíe la orden de ejecutar las acciones correspondientes. Se puede resumir todo lo dicho anteriormente en la tabla que se muestra a continuación.

Clase	Atributos	Métodos
Regla	String Descripcion List<Condicion> condiciones List<Accion> acciones	addCondicion(Condicion) addAccion(Accion) evaluar_Condiciones(Localizacion) ejecutar_Acciones()

Figura 22: Clase POI

4.2.5 Condicion

La clase Condicion.java, como se ha mencionado brevemente en la clase anterior, corresponde a las condiciones que se deben cumplir para que se ejecute una cierta acción de una cierta regla. Como las condiciones pueden ser muy diversas, se ha decidido implementar esta clase como una clase abstracta. Esto se debe a que en un futuro se pueden aumentar el número de interacciones que se pueden hacer con el framework, con lo que únicamente deberíamos añadir una clase que extienda de esta última e implemente su método comprobar. Al tratarse de una clase tan sencilla, no es necesario incluir ningún atributo en ella, bastará con crear un método abstracto que haga la correspondiente comprobación y que implementen las clases que extiendan de esta.

Clase	Atributos	Métodos
Condicion		comprobar(Localizacion)

Figura 23: Clase Condicion

4.2.6 Accion

Hay que enfatizar que esta clase está implementada como una interfaz, pues uno de los objetivos de este proyecto era no limitar las acciones que se puedan realizar con este framework. Es por esto por lo que se ha tomado la decisión de que la clase Accion sea tratada como una interfaz con un único propósito, que todas las acciones que se deseen realizar deban implementar esta interfaz y sobrescribir su único método. De este modo, cuando se decida añadir una nueva acción a realizar, lo único que se tiene que hacer es escribir dentro del método, implementado a partir de la interfaz, el código que se desea realizar cuando la condición relacionada con la regla se haya evaluado como verdadera.

Clase	Atributos	Métodos
Accion		ejecutar()

Figura 24: Clase Accion

Las siguientes clases que se pretenden mostrar son aquellas que extienden la clase Condicion. No se mostrarán las clases que implementen la interfaz Accion puesto que no se consideran clases propias de la solución, sino más bien propias de los casos de uso que se van a tratar más adelante.

4.2.7 Patrones de interacción

Seguidamente, se van a explicar cómo se ha diseñado cada uno de los patrones de interacción que se han visto en el capítulo de análisis. Se dará una explicación con el modelado ECA y posteriormente se explicará más en detalle. El modelado ECA viene definido por tres elementos, evento-condición-acción. Estos tres elementos suceden en cascada, pues la llegada el evento desencadenará la evaluación de una cierta condición y si esta se cumple se ejecutará una acción.

1. Acercarse

MODELO ECA:

EVENTO: 'Envío de posición del objeto'

CONDICIÓN: Que la posición anterior del objeto del cuál se ha recibido la posición nueva sea más lejana respecto al área (ubicación + radio) de un objeto, es decir, que la diferencia de la posición anterior menos la posición del objeto sea mayor que la diferencia de la posición actual menos la posición del objeto.

ACCIÓN: A definir dependiendo de la naturaleza de la aplicación final.

El objetivo de esta clase debe ser evaluar si una cierta localización está acercándose a un cierto POI. Como se ha comentado anteriormente, esta clase extiende la clase abstracta Condicion, por lo que debe implementar el mismo método que dicha clase. Más allá de ese método, se debe tener una variable POI propia de la clase, pues será necesario que el constructor de esta clase tenga un objeto POI, ya que será sobre la longitud y la latitud de este objeto sobre los que hará las comprobaciones necesarias junto con la longitud y latitud de la localización y devolver una evaluación.

Clase	Atributos	Métodos
Acercarse	POI poi	comprobar(Localizacion)

Figura 25: Clase Acercarse

Todo lo que se ha explicado en esta clase se puede aplicar a todas las clases posteriores, por lo que no se volverá a nombrar.

2. Alejarse

MODELO ECA:

EVENTO: 'Envío de posición del objeto'

CONDICIÓN: Que la posición anterior del objeto del cuál se ha recibido la posición nueva sea más lejana respecto al área (ubicación + radio) de un objeto, es decir, que la diferencia de la posición anterior menos la posición del objeto sea menor que la diferencia de la posición actual menos la posición del objeto.

ACCIÓN: A definir dependiendo de la naturaleza de la aplicación final.

El objetivo de esta clase es evaluar si una cierta localización está alejándose de un cierto POI.

Clase	Atributos	Métodos
Alejarse	POI poi	comprobar(Localizacion)

Figura 26: Clase Alejarse

3. Estar_Dentro

MODELO ECA:

EVENTO: 'Envío de posición del objeto'

CONDICIÓN: Que el objeto del cuál se ha recibido la posición se encuentre dentro del área (ubicación + radio) de otro objeto.

ACCIÓN: A definir dependiendo de la naturaleza de la aplicación final.

El objetivo de esta clase es evaluar si una cierta localización se encuentra dentro de un cierto POI.

Clase	Atributos	Métodos
Estar_Dentro	POI poi	comprobar(Localizacion)

Figura 27: Clase Estar_Dentro

4. Entrar

MODELO ECA:

EVENTO: 'Envío de posición del objeto'

CONDICIÓN: Que la posición anterior del objeto del cuál se ha recibido la posición nueva no se encuentre dentro del área (ubicación + radio) de un objeto y la posición actual sí que se encuentre dentro de dicha área.

ACCIÓN: A definir dependiendo de la naturaleza de la aplicación final.

El objetivo de esta clase es evaluar si una cierta localización ha entrado a un cierto POI.

Clase	Atributos	Métodos
Entrar	POI poi	comprobar(Localizacion)

Figura 28: Clase Entrar

5. Salir

MODELO ECA:

EVENTO: 'Envío de posición del objeto'

CONDICIÓN: Que la posición anterior del objeto del cuál se ha recibido la posición nueva se encuentre dentro del área (ubicación + radio) de un objeto y la posición actual no se encuentre dentro de dicha área.

ACCIÓN: A definir dependiendo de la naturaleza de la aplicación final.

El objetivo de esta clase es evaluar si una cierta localización ha salido a un cierto POI.

Clase	Atributos	Métodos
Salida	POI poi	comprobar(Localizacion)

Figura 29: Clase Salir

Finalmente, se expone una figura (figura 30) a modo resumen del framework diseñado.

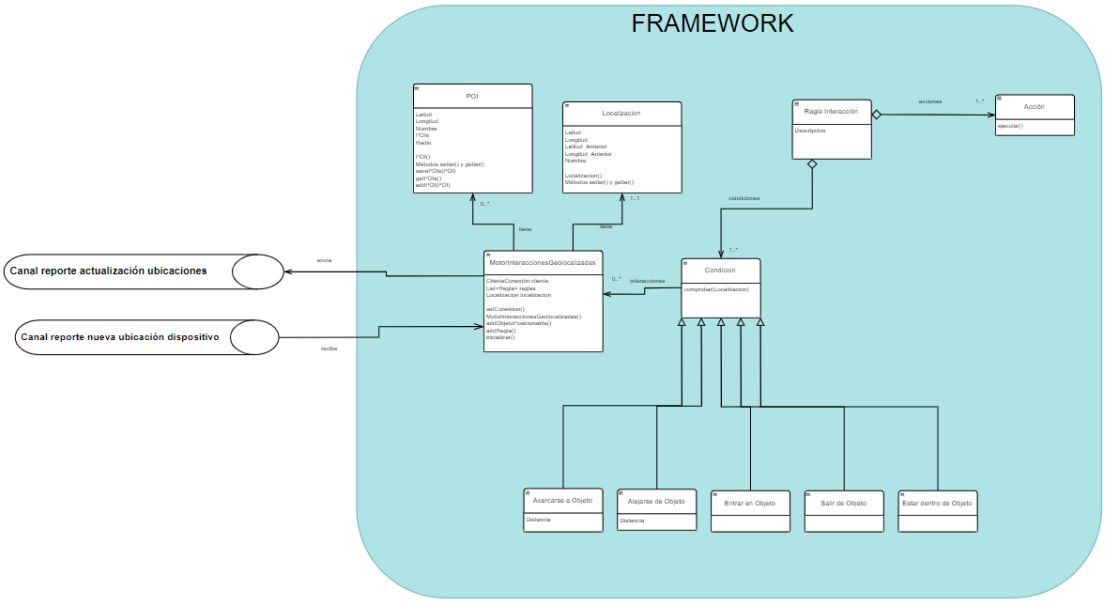

Figura 30: Resumen del framework

5. Implementación

Una vez analizados y descritos todos los requerimientos que se necesitan para dar soporte a las tareas para las que se ha propuesto esta solución, se pasará a la fase de desarrollo. Esta fase junto con la anterior son en las que más tiempo se ha invertido, ya que todo debería quedar bien definido y a su vez debería funcionar conforme se ha descrito anteriormente.

En este apartado no se pretende describir cada una de las líneas de código que han sido generadas. Lo que se quiere hacer es dar una descripción en detalle de alguno de los métodos más importantes a los que se le ha dado una implementación, así como mostrar los pasos necesarios que se han seguido hasta llegar a la solución final.

Por consiguiente, se va a dividir este apartado en dos subapartados, la creación del propio proyecto y el desarrollo de los distintos métodos más importantes del framework.

5.1 Caso de estudio

Actualmente la problemática de encontrar un framework capaz de hacer un seguimiento de la posición del usuario independiente de otra tecnología o empresa es bastante compleja. Es por esto por lo que la necesidad de crear un framework de interacciones geoespaciales se ha vuelto necesaria para poder desarrollar aplicaciones más complejas en un futuro utilizando este framework. A pesar de ello, se deben tener en cuenta una serie de requerimientos mínimos que deberán tener las aplicaciones futuras desarrolladas con esta solución.

Primeramente, se deberá entender que se depende única y exclusivamente del uso del dispositivo móvil del usuario, y de este con su módulo GPS. Esto es esencial para la futura aplicación puesto que, si por algún casual el usuario decide tener desconectado la localización móvil, la aplicación Owntracks no podrá enviar su posición en ningún caso.

La segunda de ellas es que el usuario acepte tener instalado en su dispositivo personal una aplicación de rastreo como puede ser Owntracks. En esta solución se ha planteado el uso de Owntracks, pero no sería necesario tener instalado esta aplicación específica, se podría llegar a plantear alguna alternativa.

Finalmente, relacionado con el punto anterior, se deberá plantear siempre que la aplicación alternativa funciona con el protocolo MQTT, ya que de no ser así, esta solución no sería la adecuada para una aplicación que no utilice dicho protocolo.

5.2 Tecnologías

Primeramente, se deberá hacer un estudio de la información que nos brinda cada tecnología para poder adaptarla a nuestra solución propuesta. Es por esto por lo que se ha elegido Owntracks como broker entre el dispositivo móvil y el framework. Owntracks como se ha explicado anteriormente es una aplicación móvil gratuita que funciona tanto en Android como en iOS que nos permite el envío de la ubicación del usuario. Esta herramienta utiliza el paradigma publicador/suscriptor. Este modelo propone dos aplicaciones que tienen una función muy específica. El publicador o "*publisher*" es el encargado de enviar el mensaje al servidor. Este mensaje es publicado a un cierto "*topic*". Los topics son rutas dentro del servidor en el cual se depositan los mensajes.

Basándonos en la documentación de Owntracks[5], la aplicación publica los mensajes con en una ruta por defecto "*owntracks/*". Esta ruta es personalizable, pero en este caso se ha decidido dejar esta ruta por defecto, ya que el broker MQTT[7] que se ha utilizado no implementa más soluciones cuyos mensajes pudieran llegar a interferir con los mensajes de este framework. Posteriormente a esto, para distinguir los dispositivos que utilizan la aplicación, se añade a la ruta por defecto anteriormente mencionada el propio nombre del usuario y el nombre del dispositivo móvil, teniendo así dos niveles de topics, puesto que se podría llegar a tener la casuística en la que un mismo usuario tuviese dos dispositivos móviles y esta solución seguiría siendo adecuada gracias a la alta escalabilidad. Con todo lo mencionado anteriormente, se puede concluir que la ruta completa por defecto que utiliza Owntracks es "*owntracks/<user>/<device>*", como se muestra en la figura (figura 31).

Retomando la explicación de las colas de mensajería, el dispositivo publica en la ruta "*owntracks/<user>/<device>*", esta ruta deberá ser conocida por el servidor, pues será este quien deba suscribirse a dicha ruta. Esto quiere decir que el servidor estará atento a la llegada de un mensaje enviado por el móvil desde la aplicación Owntracks. Así pues, cuando el servidor reciba el mensaje, se deberá tratar como una solicitud de procesamiento de los datos.

Una vez se han procesado los datos en el servidor, siguiendo con la problemática principal que se planteó, este deberá publicar una respuesta a otra cola de mensajería, la cual se ha considerado adecuado asemejar a la nomenclatura propuesta por Owntracks, denominándose "*app-info/<user>/<device>*". Al igual que pasaba anteriormente con la aplicación Owntracks y el servidor, será necesario configurar la aplicación que se deseé construir para que se suscriba a la ruta que se considere, y de esta forma poder recibir el mensaje que el servidor ha publicado en la cola de mensajería una vez ha analizado los datos enviados anteriormente.

[5] https://owntracks.org/booklet/

Así pues, la arquitectura implantada quedaría como se muestra en la siguiente figura.

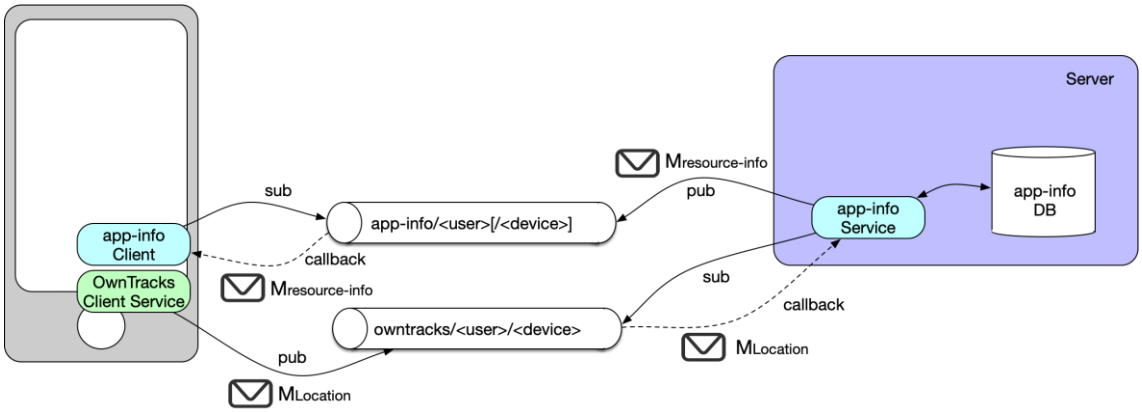

Figura 31: Arquitectura de la solución implantada [6]

5.3 Estructura del proyecto

Por lo referente a la creación del proyecto, se ha decidido implementar esta solución con un proyecto Maven. Un proyecto Maven es un tipo de proyecto que nos da una serie de beneficios los cuales ayudan a simplificar la solución final.

Una de las principales ventajas que nos ofrece este tipo de proyectos está relacionada con la gestión de las dependencias del proyecto. Cuando se crea un proyecto Maven uno de los archivos que se generan automáticamente es el "*pom.xml*". Este archivo es el encargado de descargar automáticamente cada una de las librerías necesarias para el componente a desarrollar. Por tanto y como se ha comentado anteriormente, se necesita la librería de Java Paho para realizar la conexión con el broker. La modificación que será necesaria realizar en el archivo *pom.xml* para incluir automáticamente esta librería se muestra en la siguiente imagen.

```
<dependency>
    <groupId>org.eclipse.paho</groupId>
    <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
    <version>1.2.5</version>
</dependency>
```

Figura 32: Dependencia Paho en pom.xml

Se deberá añadir las líneas mostradas en la figura anterior (figura 32) dentro de la etiqueta <*dependencies*>. De esta forma ya se podrán utilizar los métodos y las clases que nos ofrece la librería Paho, como son las siguientes clases.

- **MqttAsyncClient**: esta clase se utiliza para realizar la conexión con el broker. Se utilizan los métodos "*subscribe()*" y "*connect()*" propios de la clase. El primero de ellos se utiliza para suscribirse a un topic concreto, sobre el que se espera que el publicador deje sus mensajes. El segundo de ellos sirve para conectarse al broker utilizando la url proporcionada.

- **MqttCallback**: La interfaz MqttCallback se implementa en la clase Callback. La clase Callback es la encargada de dictaminar cual es el código que se ha de ejecutar una vez se ha recibido el mensaje del publicador. Esto se consigue gracias al método "*setCallback()*", al que se le pasa como parámetro una clase instanciada en nuestro propio proyecto con el código que queremos ejecutar.

```
public void setOwntracks(String url, String topic) {
try {
    MqttDefaultFilePersistence persistence = null;
    try {
        persistence = new MqttDefaultFilePersistence("/tmp");
    }catch (Exception e) {}
    if ( persistence != null ) {
        myClient = new MqttAsyncClient(url, UUID.randomUUID().toString(), persistence);
    }else {
        myClient = new MqttAsyncClient(url, UUID.randomUUID().toString());
    }
    Callback callback = new Callback();
    myClient.setCallback(callback);
    IMqttToken token = myClient.connect();
    token.waitForCompletion();

    myClient.subscribe(topic, 0);
 } catch (MqttException e) {
     System.out.println(e.toString());
     }

}
```

Figura 33: Clases y métodos de la librería Paho

La clase *MqttDefaultFilePersistance* se ha creado para enviar todos los archivos que se crean relacionados con cada una de las conexiones MQTT a la carpeta temporal del sistema.

```
public class Callback implements MqttCallback{

    public void connectionLost(Throwable cause) {
        System.out.println(cause.toString());
    }

    public void messageArrived(String topic, MqttMessage message) throws Exception {
        JSONObject mensaje = new JSONObject(new String(message.getPayload()));
        MotorInteraccionesGeolocalizadas.inicilizar(mensaje);
    }

    public void deliveryComplete(IMqttDeliveryToken token) {

    }
}
```

Figura 34: Método messageArrived sobrescrito

Este último método es de gran importancia también, pues la aplicación Owntracks utiliza mensajes MQTT, por lo que deberemos modificar este mensaje en una tecnología que la aplicación final entienda. Es por esto que se ha utilizado el método "JSONObject" con el mensaje MQTT [7] que recibe la aplicación. De esta manera, se transforma el mensaje en lenguaje MQTT en un mensaje de tipo JSON, el cual sí es entendible para la aplicación.

5.4 Implementación del framework

En este apartado se van a abordar los métodos más interesantes de cada clase. Por ello, se explicarán los métodos más interesantes de alguna de las clases.

5.4.1 MotorInteraccionesGeolocalizadas

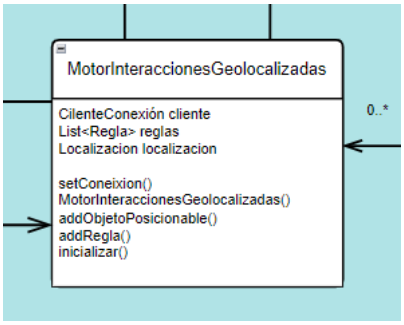

Figura 35: Clase MotorInteraccionesGeolocalizadas

Esta clase es una de las más interesantes para el framework, pues es la encargada de varias funciones muy importantes. Como se ha comentado en el apartado anterior, esta clase debe establecer la conexión con el servidor, alimentar la variable que almacena los POIs y las Reglas, y por último debe ejecutar todo el código para la evaluación de las reglas. Como se ha comentado anteriormente, esto se ha desarrollado en el método inicializar.

He de comentar que existe la posibilidad desde la aplicación Owntracks de añadir áreas independientes de las que creé el desarrollador. Es por esto por lo que se deben diferenciar los mensajes que se reciben desde la aplicación. Esto se hace gracias a que el mensaje tiene el atributo "*_type*", y a que este toma distintos valores dependiendo del tipo de mensaje. Si es una localización del usuario, en valor del atributo será "*location*". En caso de que fuera que el usuario ha añadido nuevos puntos de interés, el atributo tendría el valor "*waypoints*" en caso de que optara por publicar varias áreas a la vez o "*waypoint*" en caso de que solo fuera un área. Alegar que estas áreas, a pesar de tenerlas en la variable interna de áreas, al no estar ligada a una regla, no se

tendrán en consideración para las evaluaciones. A continuación, se ilustra el código del método inicializar ya que se considera de gran importancia.

```
public static void inicilizar(JSONObject mensaje) {
    String tipo = mensaje.getString("_type");
    if(tipo.equals("location")) {
        if(localizacion.getLatitud() == null) {
            Double latitud = mensaje.getDouble("lat");
            Double longitud = mensaje.getDouble("lon");
            localizacion = new Localizacion(latitud,null,longitud,null,"GPS");
            for(Regla regla : reglas) {
                if(regla.evaluar_Condicion(localizacion)) {
                    regla.ejecutar_Accion();
                };
            }
        }else {
            Double latitudNueva = mensaje.getDouble("lat");
            Double longitudNueva = mensaje.getDouble("lon");
            localizacion.setLatitud(latitudNueva);
            localizacion.setLongitud(longitudNueva);
            for(Regla regla : reglas) {
                if(regla.evaluar_Condicion(localizacion)) {
                    regla.ejecutar_Accion();
                };
            }
            localizacion.setLatitudAnterior(latitudNueva);
            localizacion.setLongitudAnterior(longitudNueva);
        }
    }else{
        if(tipo.equals("waypoints")) {
        POI.retrievePOIs(mensaje);
        }else {
            if(tipo.equals("waypoint")) {
            POI.addPOIJSON(mensaje);
            }
        }
    }
```

Figura 36: Método inicializar

El método de conexión con el servidor es el que se ha mostrado antes. El resto de los métodos no se consideran de una gran complejidad por lo que no se mostrarán.

5.4.2 Localización

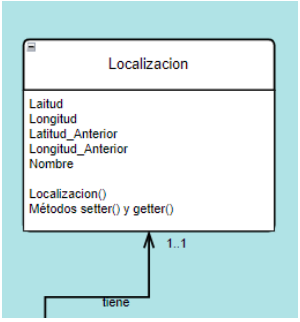

Figura 37: Clase Localizacion

La clase localización es la que se encarga de guardar la localización del propio usuario. Anteriormente se han comentado los métodos que debe tener esta clase, de los cuáles no se considera de gran relevancia ninguno de ellos pues únicamente son métodos setter y getter habituales.

5.4.3 POI

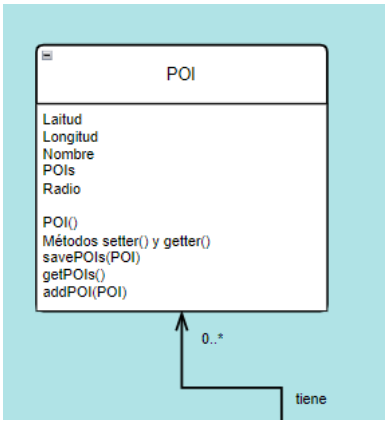

Figura 38: Clase POI

El objetivo principal de esta clase era representar las áreas del espacio sobre las cuales se quieren evaluar las distintas reglas que se han establecido previamente. Para ello se deberían implementar los métodos setter y getter de cada uno de los atributos propios de la clase. Estos métodos no se consideran relevantes para la solución puesto que son muy triviales. Sin embargo, los métodos "*retrievePOIs(JSONObject)*", "*addPOIJSON(JSONObject)*" y "*savePOIs*" sí que son de alto interés para la solución.

Los métodos retrievePOIs y addPOIJSON actúan de manera similar. Ambos llaman al método savePOIs para guardar los POIs o áreas que ha establecido el usuario. La diferencia principal entre uno y otro es la cantidad de áreas que se guardan en el atributo POIs que guarda las áreas. El primero de ellos, retrievePOIs, almacena todos los POIs que ha establecido el usuario con la opción de Owntracks de "*Publicar Waypoints*". Por otra parte, el método addPOIJSON almacena únicamente un área definida por el usuario cuando la crea en la propia aplicación.

```
public static void retrievePOIs(JSONObject mensaje) {

    int longitud_waypoints = mensaje.getJSONArray("waypoints").length();

    for(int i = 0; i < longitud_waypoints; i++) {
        Double longitudPOI = mensaje.getJSONArray("waypoints").getJSONObject(i).getDouble("lon");
        Double latitudPOI = mensaje.getJSONArray("waypoints").getJSONObject(i).getDouble("lat");
        Double radioPOI = mensaje.getJSONArray("waypoints").getJSONObject(i).getDouble("rad");
        String nombrePOI = mensaje.getJSONArray("waypoints").getJSONObject(i).getString("desc");
        POI poi = new POI(latitudPOI,longitudPOI,nombrePOI,radioPOI);
        poi.savePOIs(poi);
    }
    System.out.println("Waypoints publicados!");
}
```

Figura 39: Método retrievePOIs

```
public static void addPOIJSON(JSONObject mensaje) {

    Double longitudPOI = mensaje.getDouble("lon");
    Double latitudPOI = mensaje.getDouble("lat");
    Double radioPOI = mensaje.getDouble("rad");
    String nombrePOI = mensaje.getString("desc");
    POI poi = new POI(latitudPOI,longitudPOI,nombrePOI,radioPOI);
    poi.savePOIs(poi);
}
```

Figura 40: Método addPOIJSON

Como se muestra en la figura 39, la primera tarea que debe realizar el método retrievePOIs es recoger la longitud del objeto JSON que ha recibido como argumento en su respectiva llamada. Esto se hace para posteriormente iterar en cada uno de los elementos de ese mensaje, y de esta forma guardar la longitud, la latitud, el radio y la descripción que ha indicado el usuario en la aplicación de Owntracks. Después de que se tengan toda esta información, se crea un nuevo POI con los datos indicados y se llama al método savePOIs para almacenar todas las áreas en la variable correspondiente como se mostrará con la explicación del método savePOIs. El método addPOIJSON hace exactamente lo mismo que el método anterior, salvando que este al recoger la información de un único POI no es necesario iterar el mensaje que recibe como argumento, pues únicamente va a haber un registro.

Se debe tener en cuenta una clara desventaja que tiene la aplicación Owntracks, la eliminación por parte del usuario de las áreas que el mismo ha definido. Así como con el resto de las acciones que puede hacer el usuario la aplicación sí envía un mensaje MQTT [7] distinto, a la hora de gestionar la eliminación de las áreas no funciona de la misma manera. Owntracks no envía un mensaje de eliminación de un área específica que ha declarado el usuario desde la propia aplicación. Es por esto por lo que se deberá tener alguna forma de eliminar las posibles áreas que defina el usuario, siempre y cuando la solución a la que se quiera llegar tenga la necesidad de utilizar las áreas definidas por el usuario en la aplicación. Como apunte, si el desarrollador final únicamente quiere tener en cuenta las áreas definidas por los usuarios, se podría forzar el borrado del

atributo POIs de la clase en cada ejecución del método, como se muestra a continuación.

```
public static void retrievePOIs(JSONObject mensaje) {

    /*FORZAMOS A QUE SE SOBREESCRIBA EL ARRAY YA QUE AL ELIMINAR UNA ZONA NO SE ENVIA NIGUN MENSAJE*/

    POIs = new POI[0];

    int longitud_waypoints = mensaje.getJSONArray("waypoints").length();

    for(int i = 0; i < longitud_waypoints; i++) {
        Double longitudPOI = mensaje.getJSONArray("waypoints").getJSONObject(i).getDouble("lon");
```

Figura 41: Borrado POIs

De esta forma, si el usuario decide borrar un área y posteriormente publicar todas las que tiene definida, tendremos una foto más actualizada y real de las áreas. Clarificar que esto no es una solución óptima, puesto que se depende de que el usuario haga una acción, lo cual puede ocurrir o no.

Por último, el método savePOIs es el encargado de guardar en la variable POIs los POIs que se han construido en las llamadas a los métodos anteriores. Para optimizar el código, se ha decidido que únicamente se puedan guardar áreas que tienen un nombre distinto, pues se ha supuesto que si tienen el mismo nombre es porque tienen la misma latitud y longitud.

```
public void savePOIs(POI nuevoPOI) {
    if(POIs.length == 0) {
        POI[] newPOIs = new POI[1];
        newPOIs[0] = nuevoPOI;
        POIs = newPOIs;
    }else {
        int longitudArray = POIs.length;
        boolean encontrado = false;
        for(int k = 0; k < longitudArray; k++) {
            if(nuevoPOI.nombre.equals(POIs[k].nombre)) {
                encontrado = true;
            }
        }
        if(!encontrado) {
        POI[] newPOIs = new POI[longitudArray+1];
        for(int i = 0; i<longitudArray;i++) {
            newPOIs[i] = POIs[i];
            }
        newPOIs[longitudArray] = nuevoPOI;
        POIs = newPOIs;
        }
        }
    }
```

Figura 42: Método savePOIs

Como se ve en la figura anterior (figura 42), lo primero que se comprueba en el método savePOIs es la longitud del array que guarda todas las áreas, puesto que,

si en esta variable aún no hay ningún elemento guardado, lo que deberemos hacer es crear un nuevo array con una única posición, guardar ahí el POI que se indique. Luego se deberá sobrescribir la variable POIs con el nuevo array para así no perder ese registro añadido. Sin embargo, si la variable POIs ya tiene un registro como mínimo, lo que se deberá hacer es iterar por todo el array buscando si en alguna de sus posiciones existe un POI con un nombre igual al que se quiere insertar. De no ser así, entonces se crea un nuevo array con capacidad n+1, siendo n el número de registros que tiene el array actualmente, se traspasan todos los datos de el array anterior al nuevo y se añade en el último registro el POI nuevo que se quiere insertar. Por último, se vuelve a sobrescribir el valor de POIs con el nuevo array.

Finalmente, esta clase tiene dos métodos addPOI y getPOIs los cuales hacen que se añada un nuevo POI a la variable POIs llamando al método savePOIs, y devolver los valores almacenados en la variable POIs respectivamente.

```
public static void addPOI(POI poi) {
    poi.savePOIs(poi);
}
public static POI[] getPOIs() {
    return POIs;
}
```

Figura 43: Métodos addPOI y getPOIs

5.4.4 Regla

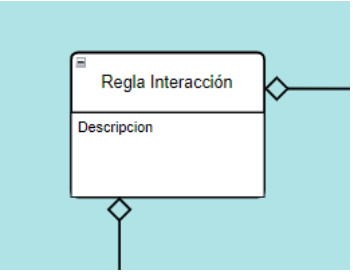

Figura 44: Clase Regla

Por lo referente al código de esta clase, llama la atención, pero no por su complejidad sino por la funcionalidad que se le ha querido dar. La clase regla relaciona la clase condición y la clase acción. Por ello tiene los métodos addCondicion y addAccion. El método addCondicion tiene la funcionalidad de guardar cada una de las condiciones que se quieren evaluar para determinar si una cierta regla se ha cumplido. Por otra parte, el método addAccion almacena todas las acciones que deben ocurrir si una cierta regla es evaluada como verdadera. A su vez, a parte de esos dos métodos, también se han implementado los métodos para evaluar las condiciones de una regla concreta y el de ejecutar las acciones correspondientes de dicha regla.

```java
public class Regla {
    private String descripcion;
    private List<Condicion> condiciones;
    private List<Accion> acciones;

    public Regla(String descripcion) {
        this.descripcion = descripcion;
        condiciones = new ArrayList<Condicion>();
        acciones = new ArrayList<Accion>();
    };

    public void addCondicion(Condicion condicion) {
        this.condiciones.add(condicion);
    }
    public void addAccion(Accion accion) {
        this.acciones.add(accion);
    };

    public boolean evaluar_Condiciones(Localizacion localizacion) {
        boolean verdad = true;
        for (Condicion condicion : condiciones) {
            if(condicion.comprobar(localizacion) && verdad) {
                verdad = true;
            }else{
                verdad = false;
                };
        }
        return verdad;
    }

    public void ejecutar_Acciones() {
        for(Accion accion : acciones) {accion.ejecutar();}

    };
```

Figura 45: Código de la clase Regla

5.4.5 Condicion

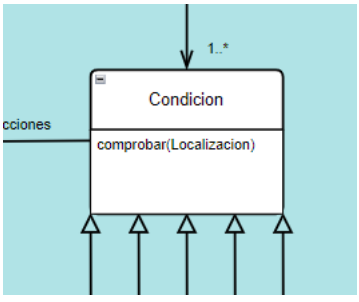

Figura 46: Clase Condicion

Para esta clase se va a utilizar un tipo de clase específica que nos brinda la tecnología Java, las clases abstractas. Una clase abstracta sirve para crear subclases de esta que sobrescriban los métodos que la clase padre tenga como "*abstract*". De esta forma, tendremos la clase Condicion que nos servirá como una clase genérica de "interacciones" y todas las subclases que hereden de ella

serán distintos tipos de interacciones que sobrescribirán el método “comprobar” acorde a las necesidades de cada una de las interacciones propuestas.

```
public abstract class Condicion {

    public Condicion() {};

    public abstract boolean comprobar(Localizacion localizacion);

}
```

Figura 47: Código de la clase Condicion

5.4.6 Accion

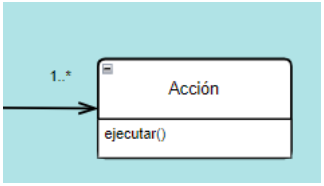

Figura 48: Clase Accion

Al igual que la clase Condicion, la clase Acción no es compleja a nivel de código. Esta clase se ha implementado como una interfaz que sirva para dejar libertad al desarrollador en un futuro de decidir qué es lo que se quiere hacer una vez se ha evaluado una condición de una regla como verdadera.

```
public interface Accion {

    void ejecutar();

}
```

Figura 49: Código de la clase Accion

5.4.7 Patrones de interacción

A continuación, se van a exponer como se han implementado cada una de las subclases que heredan de la clase Condicion, como se muestra en la siguiente figura (figura 50).

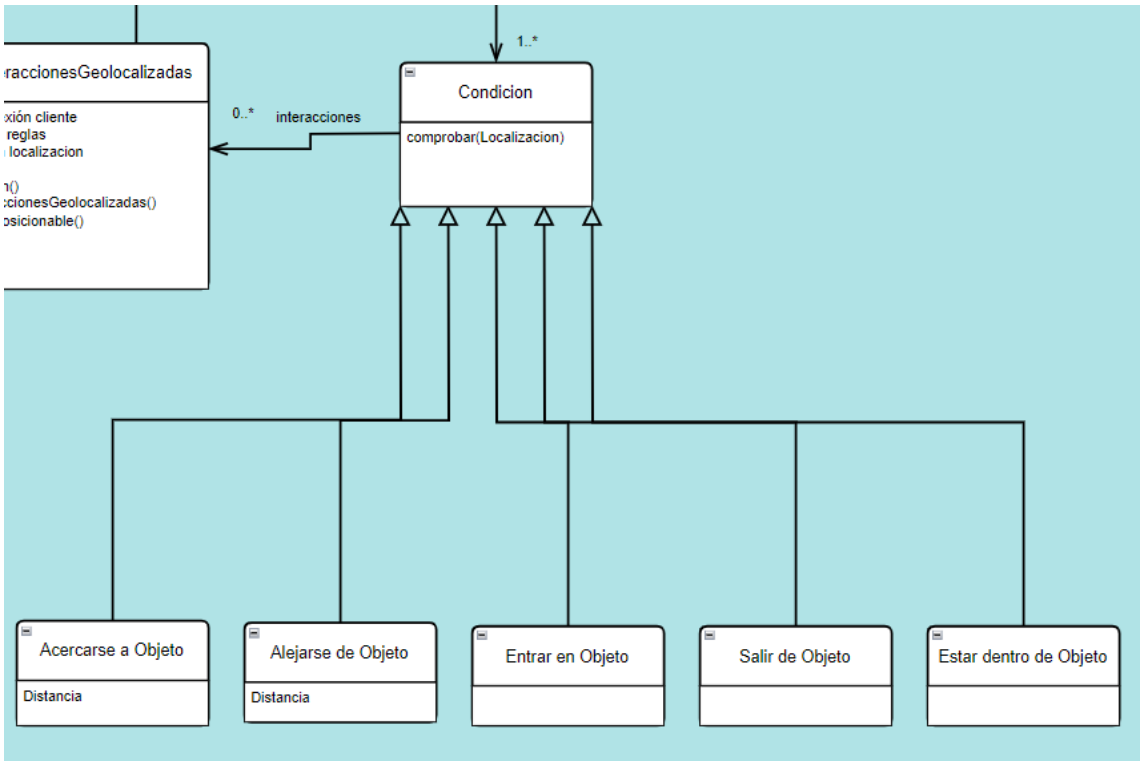

Figura 50: Subclases de Condicion en el esquema

1. Acercarse

La clase Acercarse es una clase del conjunto de subclases que extienden de la clase Condicion, y que por tanto, deben implementar el método comprobar. Así pues, se ha hecho un estudio acerca de como se podría evaluar si una localización está o no acercándose o alejándose de una cierta área. Con todo ello, se ha sobrescrito el método conforme a este estudio.

```
public class Acercarse extends Condicion{

    private POI poi;

    public Acercarse(POI poi) {
        this.poi = poi;
    };
    public boolean comprobar(Localizacion localizacion) {
        boolean respuesta = false;
        try {
        Double latitudAnterior_GPS = localizacion.getLatitudAnterior();
        Double longitudAnterior_GPS = localizacion.getLongitudAnterior();
        Double latitud_GPS = localizacion.getLatitud();
        Double longitud_GPS = localizacion.getLongitud();
        Double latitud_POI = poi.getLatitud();
        Double longitud_POI = poi.getLongitud();
        Double radio_POI = poi.getRadio();

        Double distancia_anterior = Math.sqrt((Math.pow((latitud_POI - latitudAnterior_GPS), 2) + Math.pow((longitud_POI - longitudAnterior_GPS), 2))) - radio_POI;
        Double distancia_nueva = Math.sqrt((Math.pow((latitud_POI - latitud_GPS), 2) + Math.pow((longitud_POI - longitud_GPS), 2))) - radio_POI;

        if(distancia_anterior >= distancia_nueva) {
            respuesta = false;
         }else {
             respuesta = true;
         }
        }catch(Exception e) {
        }
        return respuesta;
    }
}
```

Figura 51: Clase Acercarse

Como se muestra en la figura (figura 51), uno de los primeros pasos que se deberá hacer es recoger los datos necesarios de la localización para

posteriormente aplicarlos a la fórmula euclidiana[6] un poco modificada. La forma euclidiana es la siguiente.

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Figura 52: Forma euclidiana punto a punto 2D

Esta fórmula calcula la distancia "ordinaria" entre dos puntos en un espacio 2D, basada en el teorema de Pitágoras. Pero en el caso que se propone no sería del todo cierto pues no es la distancia respecto dos puntos, sino la distancia de un punto respecto a un área. De esta forma, una vez se han calculado tanto la distancia de la localización anterior respecto al centro del área, como la distancia de la localización actual respecto al centro del área, se debe restar el radio a ambas distancias para saber realmente cual es la distancia desde la localización hasta el punto más cercano del área. Por último, justo después de calcular las distancias, se deberán comparar entre ellas, puesto que si la actual es menor que la antigua se ha acercado y de no ser estrictamente así se estaría alejando.

2. Alejarse

Esta clase está estrictamente ligada a la anterior, pues la fórmula de calcular la distancia es exactamente la misma. La diferencia fundamental reside en la comprobación, ya que se hará al revés, en este caso, cuando la distancia anterior sea menor que la distancia actual deberá evaluarse como verdadera. Esto se consigue invirtiendo los valores asignados a la variable respuesta en la comprobación.

3. Estar_Dentro

La clase Estar_Dentro también forma parte del conjunto de subclases que extienden de la clase Condicion. Así pues, se ha sobrescrito el código del método comprobar con la implementación que se ha considerado adecuada tras hacer un estudio acerca de cuando se debe considerar que un usuario está o no dentro de un área.

[6] https://es.wikipedia.org/wiki/Distancia_euclidiana

```java
public class Estar_Dentro extends Condicion{

    private POI poi;

    public Estar_Dentro(POI poi) {
        this.poi = poi;
    };
    public boolean comprobar(Localizacion localizacion) {
        boolean respuesta = false;
        try {
        Double latitud_GPS = localizacion.getLatitud();
        Double longitud_GPS = localizacion.getLongitud();
        Double latitud_POI = poi.getLatitud();
        Double longitud_POI = poi.getLongitud();
        Double radioPOI = poi.getRadio();
        if(latitud_GPS > (latitud_POI - radioPOI) && latitud_GPS < (latitud_POI + radioPOI) &&
                longitud_GPS > (longitud_POI - radioPOI) && longitud_GPS < (longitud_POI + radioPOI)) {
            respuesta = true;
        }else {
            respuesta = false;
        }
        }catch(Exception e) {}
        return respuesta;
    }
}
```

Figura 53: Clase Estar_Dentro

Lo más importante de este método es la forma de comprobar si la localización que ha recibido se encuentra en el POI con el que se ha creado la condición. Esta fórmula comprueba que la latitud enviada por el móvil del usuario se encuentre entre la latitud con la que se ha creado el POI menos el radio y la latitud más el radio. Básicamente se debe comprobar que el valor que recibimos se encuentre entre un valor máximo y un valor mínimo, para de esta forma asegurarnos que está dentro del área. Ocurre lo mismo con la longitud, se deberá comprobar si el valor recibido se encuentra entre un valor máximo y un valor mínimo para justificar que el usuario se encuentra dentro del área. Evidentemente se deben cumplir estos dos requerimientos tanto para la latitud como para la longitud.

4. Entrar

Al igual que todas las anteriores clases, esta clase representa otra interacción geoespacial. Por ende, se ha hecho una implementación personalizada del método comprobar para evaluar cuando un usuario entra a un área.

```
public class Entrar extends Condicion{

    private POI poi;

    public Entrar(POI poi) {
        this.poi = poi;
    };

    public boolean comprobar(Localizacion localizacion) {
        boolean respuesta = false;
        try {
            Double latitudAnterior_GPS = localizacion.getLatitudAnterior();
            Double longitudAnterior_GPS = localizacion.getLongitudAnterior();
            Double latitud_GPS = localizacion.getLatitud();
            Double longitud_GPS = localizacion.getLongitud();
            Double latitud_POI = poi.getLatitud();
            Double longitud_POI = poi.getLongitud();
            Double radioPOI = poi.getRadio();

            if(latitudAnterior_GPS > (latitud_POI - radioPOI) && latitudAnterior_GPS < (latitud_POI + radioPOI) &&
                    longitudAnterior_GPS > (longitud_POI - radioPOI) && longitudAnterior_GPS < (longitud_POI + radioPOI)) {
                if(latitud_GPS > (latitud_POI - radioPOI) && latitud_GPS < (latitud_POI + radioPOI) && longitud_GPS > (longitud_POI - radioPOI) &&
                        longitud_GPS < (longitud_POI + radioPOI)) {
                    respuesta = false;
                }else {
                    respuesta = true;
                }}else {
                    respuesta = false;
                }
            }catch(Exception e) {
                return false;
            }
        return respuesta;
    }
}
```

Figura 54: Clase Entrar

Como se aprecia, lo interesante de todos los métodos anteriores radica en la comprobación de las latitudes y longitudes de cada elemento. En este caso, se añade la dificultad de tener que depender de una comprobación previa para dar una respuesta futura. Explicando la comprobación, primeramente, se deberá comprobar si la posición anterior del usuario se encuentra dentro o fuera del área específica. Si se encuentra dentro quiere decir que el usuario no ha entrado al área, por lo que se evaluará la condición como falsa. Por otro lado, si la posición anterior se encuentra fuera, se ha de comprobar nuevamente si la posición del usuario se encuentra dentro o fuera, pero en este caso se deberá comprobar si la actual. Con todo ello, si la posición actual se encuentra dentro quiere decir que el usuario ha entrado y por tanto se evaluará la condición como afirmativa. En caso contrario lo que nos indica es que el usuario se encontraba fuera del área y sigue siendo así, por lo que devolverá una evaluación negativa.

5. Salir

Al igual que ocurría con las clases alejarse y acercarse, las clases salir y entrar están estrictamente ligadas por ser interacciones contradictorias una con otra. En este caso, la primera comprobación que se hace es verificar si la posición anterior del usuario se encontraba dentro del área definida. Si se encuentra dentro se evalúa la siguiente condición que comprueba si actualmente el usuario está fuera del área. De ser así, se evaluaría la condición como verdadera, en cualquier otro caso la evaluación es falsa.

6. Ejemplo de uso

En este penúltimo apartado se pretende mostrar al lector una serie de casos de uso en los que se podría implementar la solución propuesta [12]. Para ello se ha hecho una investigación de las necesidades actuales de los usuarios y se han encontrado cuatro casos de uso en los que se podrían desarrollar soluciones interesantes con el framework propuesto. Destacar que en todo momento se asume que el usuario lleva encima su dispositivo móvil. [8]

6.1 Casos de uso

Caso de uso 1 – Ofertas centro comercial.

Un usuario está acercándose a un centro comercial registrado con un POI en nuestra aplicación. La aplicación Owntracks manda un mensaje MQTT con las coordenadas del usuario. El motor de interacciones geoespaciales analiza esos datos y se da cuenta que asociado a ese POI existe una regla que se cumple, por lo que ejecuta su acción. La acción en este caso sería hacer una llamada REST a una base de datos para recoger las ofertas de las tiendas de dicho centro comercial.

Caso de uso 2 – Asistente turístico

Un usuario está dentro de una ciudad registrada con un POI en nuestra aplicación. La aplicación Owntracks envía un mensaje MQTT al servidor con las coordenadas del usuario. Este último analiza los datos y los procesa. Al procesarlos una regla asociada a ese POI se cumple, por lo que el motor ejecuta la acción. En este caso sería mostrar un resumen de la ciudad visitada por el turista y sus monumentos emblemáticos.

Caso de uso 3 – Smart home

Un usuario está alejándose de su casa, la cual tiene registrada en la aplicación con un POI. Owntracks envía un mensaje al servidor con las coordenadas del usuario y el motor de interacciones evalúa los datos recibidos. La evaluación de estos datos finaliza y da como resultado que una regla sobre ese POI es verdadera, por lo que el motor ejecuta la acción asociada. En este caso será un resumen del funcionamiento y estado de todos los electrodomésticos IoT de la casa, con la opción de cambiar alguno de estos dos.

Caso de uso 4 – Control parental

Un estudiante ha salido de la universidad camino a su casa, tiene registrada la universidad como un POI en nuestra aplicación. En cierto momento la aplicación Owntracks envía un mensaje con las coordenadas del usuario al servidor, el cual está ejecutando nuestro motor de acciones geoespaciales. Cuando el motor acaba de evaluar los datos recibidos por la aplicación, comprueba que hay una regla que se cumple asociada al evento de que el usuario salga de la universidad y ejecuta la correspondiente acción. En este caso sería enviar una notificación push a los teléfonos asociados para recibir dicha notificación, mostrando en la misma que el estudiante ha salido de la universidad.

De este modo, una vez expuestos los distintos casos de uso, se pretenden desarrollar diferentes interacciones en dos de los escenarios propuestos.

6.1.1 Asistente turístico

Interacción 1 – Entrar Alzira

El usuario entra dentro del rango que tenemos declarado en nuestra aplicación como “Alzira”, por lo que Owntracks envía el mensaje, lo que el servidor lo recoge y evalúa las coordenadas. La evaluación da como conclusión que el usuario ha entrado dentro del rango, por lo que la regla asociada a esa interacción se evalúa como verdadera y se ejecuta la acción. En este caso se enviaría al usuario un resumen de la historia de Alzira y sus puntos de interés.

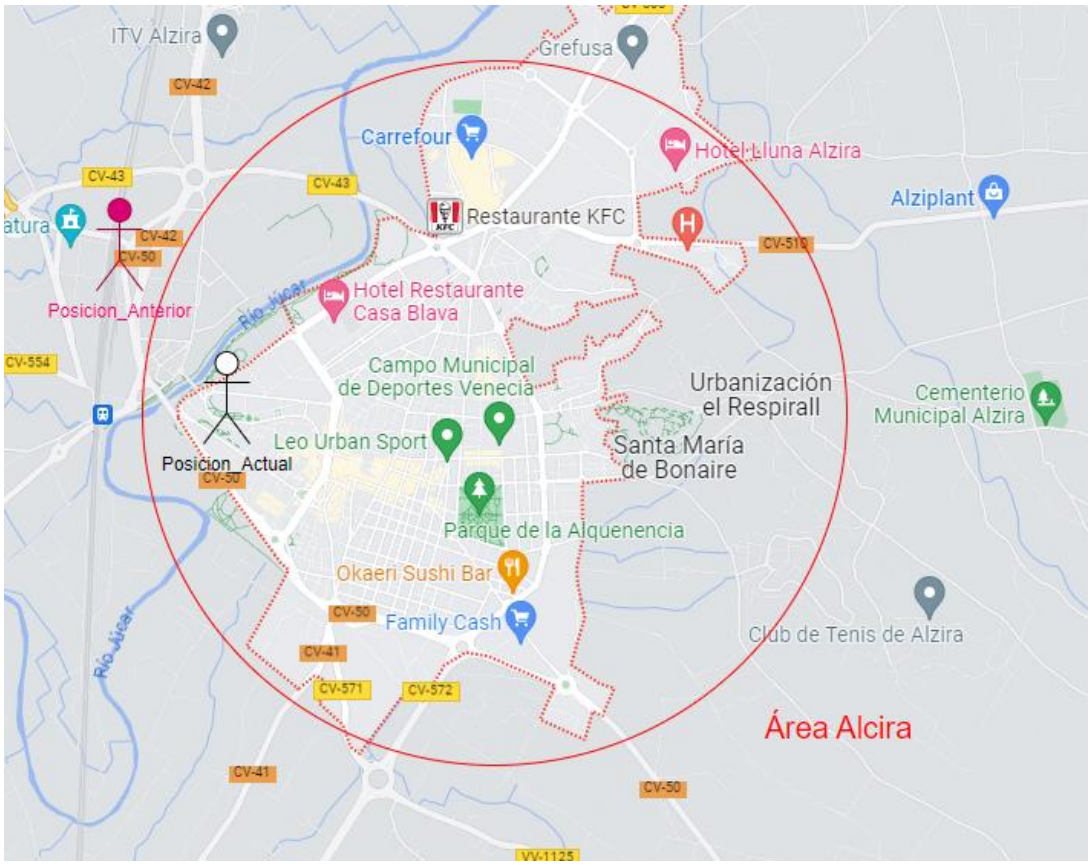

Figura 55: Interacción Entrar Alzira

Interacción 2 – Acercase al ayuntamiento.

El usuario se está acercando al ayuntamiento de Alzira, por lo que cuando la aplicación recibe las coordenadas y se da cuenta de ello, la regla “acercarse al ayuntamiento” se cumple por lo que se ejecuta la acción correspondiente. Esta acción pretende dar a conocer el partido actual que gobierna en Alzira y la distancia hacía los lugares emblemáticos.

Figura 56: Interacción Acercarse Ayuntamiento

Interacción 3 – Estar en el Museo

El usuario se encuentra dentro del museo de Alzira. Owntracks envía un mensaje al servidor, lo que este lo recoge y procesa. Tras procesarlo, se cumple la regla de estar dentro del museo, por lo que se ejecuta su acción. En este caso se pretende dar a conocer al usuario las diferentes exposiciones que habrá en los próximos días, así como un breve resumen de cada una de ellas.

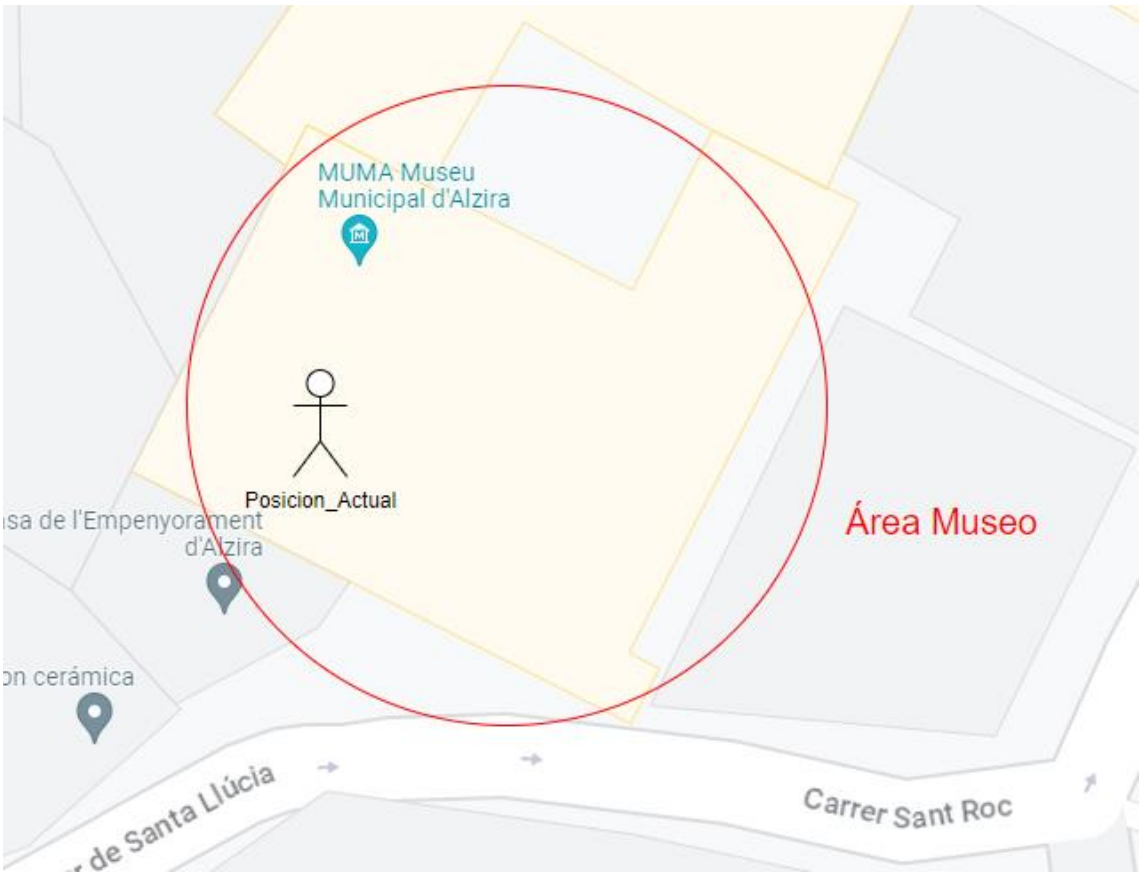

Figura 57: Interacción Estar en el museo

Interacción 4 – Acercarse al parque de la Alquencia

Owntracks envía un mensaje MQTT al servidor con las coordenadas del usuario. Al procesar dicho mensaje, el servidor se da cuenta que el usuario se está acercando al punto de interés registrado como “Parque Alquencia”, el cual tiene asociada una regla de proximidad, por lo que esta se cumple y se ejecuta. La acción pretende dar a conocer todos los monumentos que se encuentran dentro del parque y una ruta guiada que pasa por todos ellos.

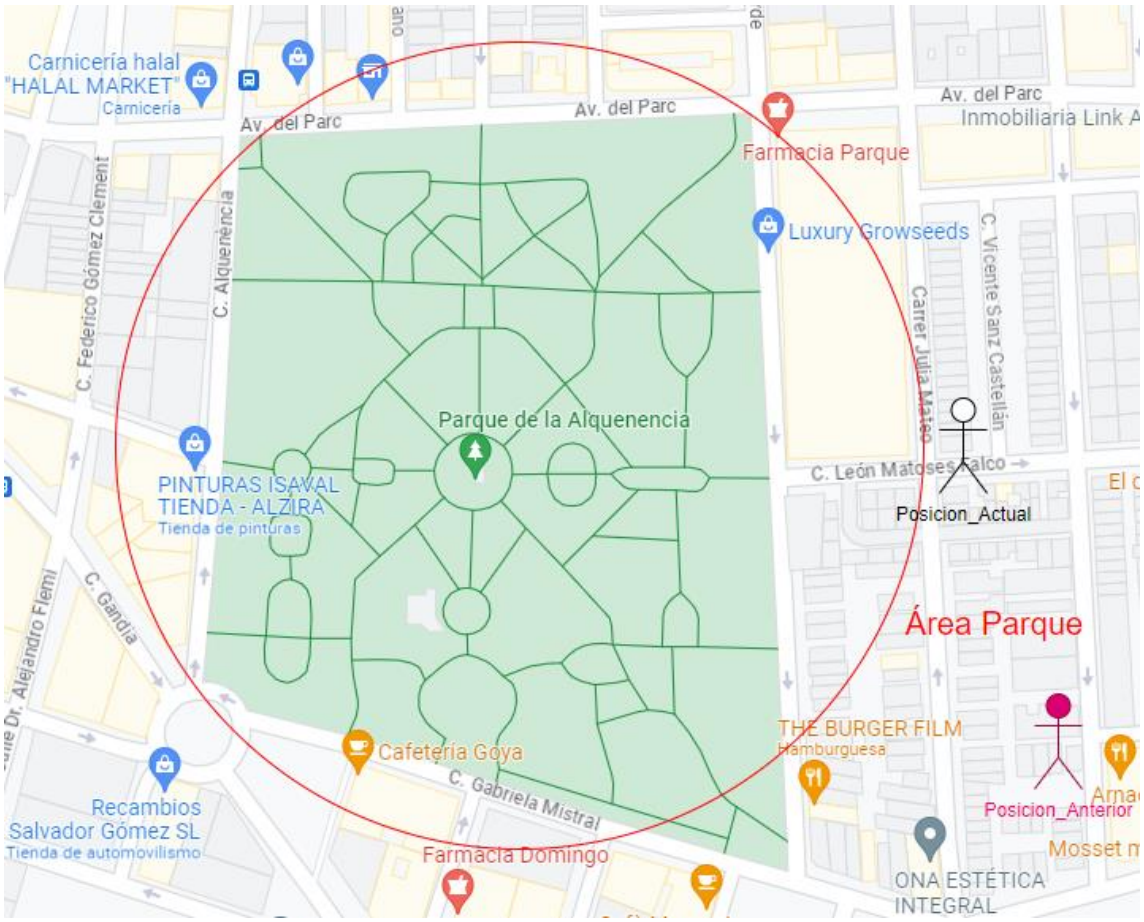

Figura 58: Interacción Acercarse Parque de la Alquencia

Interacción 5 – Salir de Alcira

El usuario acaba de salir del rango de Alzira. Owntracks envía un mensaje al servidor y este lo evalúa. Al hacer esto, se da cuenta que el usuario cumple la condición de la regla “Salir de Alzira”, por lo que se ejecuta su acción. Esta acción muestra los lugares visitados dentro de Alzira y da la posibilidad de puntuarlos.

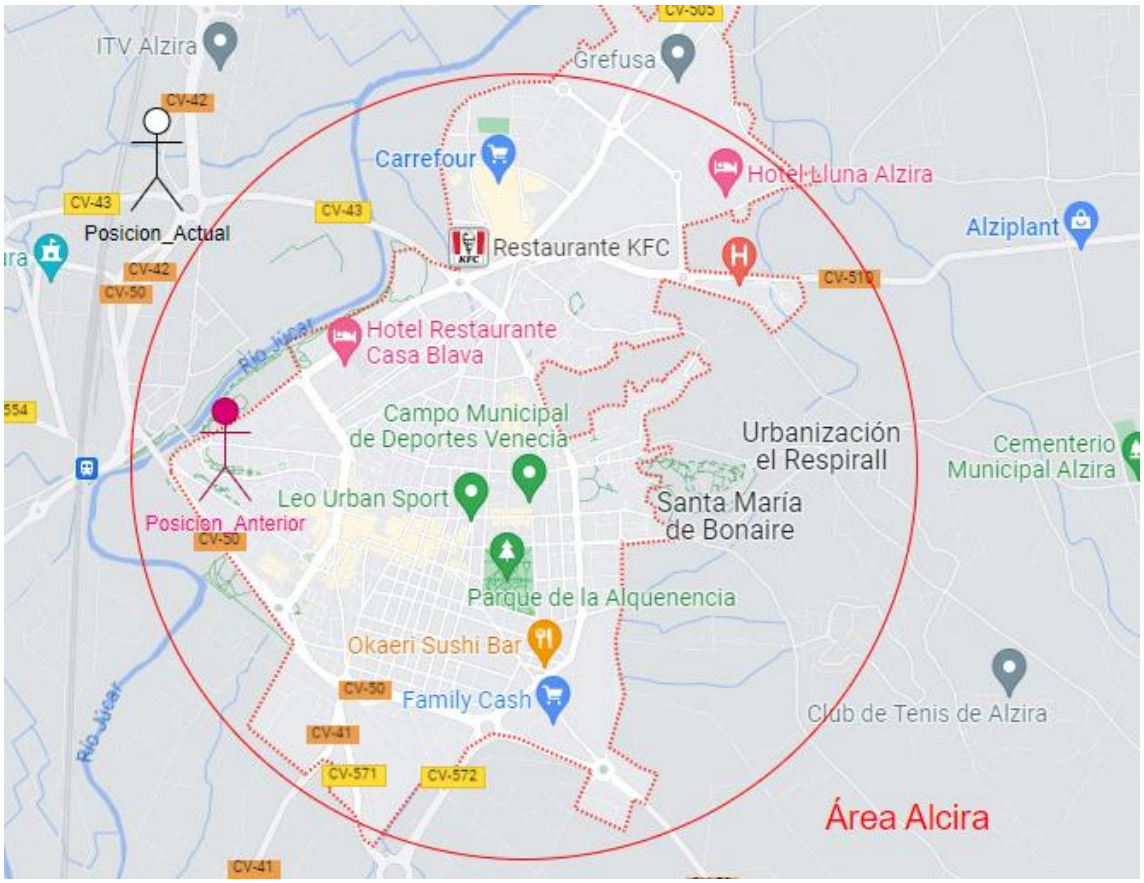

Figura 59: Interacción Salir de Alcira

6.1.2 Smart home

Interacción 1 – Alejarse de la casa

Un usuario está alejándose de su casa, la cual tiene registrada en la aplicación con un POI. Owntracks envía un mensaje al servidor con las coordenadas del usuario y el motor de interacciones evalúa los datos recibidos. La evaluación de estos datos finaliza y da como resultado que una regla sobre ese POI es verdadera, por lo que el motor ejecuta la acción asociada. En este caso será un resumen del funcionamiento y estado de todos los electrodomésticos IoT de la casa, con la opción de cambiar alguno de estos dos.

Interacción 2 – Estar en la nevera

Un usuario se encuentra dentro de su casa donde tiene definida la nevera como un punto de interés. La aplicación Owntracks envía un mensaje al servidor y este lo procesa. Al procesarlo se da cuenta que el usuario se encuentra al lado de la nevera y este punto de interés tiene asociado una regla de “estar cerca” por lo que se cumple dicha regla y se ejecuta su acción. La acción muestra una lista de

la compra que comparten todos los miembros de la familia, con la posibilidad de añadir un nuevo producto.

Interacción 3 – Acercarse a la casa

Un usuario se está acercando a su casa con su coche. La aplicación Owntracks envía al servidor un mensaje con las coordenadas de este. Cuando el servidor procesa estos datos, se cumple la regla de acercarse a la casa, por lo que se ejecuta la correspondiente acción. En este caso, se envía una señal para que la puerta principal del garaje de la casa se abra.

Interacción 4 – Entrar en casa

Un usuario está entrando por la puerta de su casa y tiene instalado la aplicación Owntracks. Esta aplicación envía un mensaje MQTT al servidor para que este lo procese. Al procesarlo se cumple la condición de la regla “Entrar en casa” por lo que se ejecuta la acción. Se mostrará al usuario el estado de los dispositivos IoT y la posibilidad de cambiarlos.

7. Conclusiones

Finalmente, implantadas todas las interacciones que se han considerado de mayor relevancia hecho el análisis de los usuarios y redactando en detalle alguno de los casos de uso que se han analizado, podemos extraer una serie de conclusiones.

La problemática que se presentó en un principio como una necesidad por parte del alumnado de investigar acerca de las interacciones geoespaciales, como funcionaban, que había detrás de ellas, como era posible que mediante un dispositivo móvil pudieran surgir tanta información ha sido el desencadenante de este proyecto. Después de mucho esfuerzo y trabajo, el proyecto ha sido realizado con cada uno de los objetivos iniciales que se planteó, siendo uno de los más importantes la necesidad de trabajar independiente a otras tecnologías y empresas que pudiesen reclamar un uso indebido de su tecnología.

Por otra parte, se ha dejado total libertad al desarrollador futuro para escoger que es lo que se debe hacer una vez se ha completado la evaluación de una regla y se debe ejecutar una acción, pues la condición es verdadera. No se ha limitado en ningún momento a implementar una tecnología aparte para cuando esto ocurra ni se han limitado las opciones de las que dispone el desarrollador, pues se ha considerado esto como una base sobre la cual se deben construir soluciones a problemas más concretos y no algo tan general, pues si se centraba en lo concreto no se hubiese podido extrapolar a otros ambientes y la solución no hubiera funcionado en todos los escenarios posibles imaginables.

Así pues, se ha entendido muy bien cómo funcionan las comunicaciones con el protocolo de mensajería MQTT, que es un broker y el paradigma publicador suscritor que es con el que se ha trabajado durante todas las fases del proyecto.

7.1 Trabajos Futuros

Este mundo de las tecnologías IoT es muy denso, por lo que se ha quedado alguna cosa pendiente en la realización de este proyecto.

Se podría hacer un estudio acerca de más interacciones geoespaciales, no se ha limitado el framework solo a las que se han mostrado, sino que tiene una alta escalabilidad, pudiendo hacerse versiones posteriores en las cuales se incluyan nuevas formas de relacionarse con el mundo y con el resto de los dispositivos móviles e IoT. No tenemos que regirnos únicamente por los móviles, hay muchos dispositivos de los que se podría extraer más información, como podrían ser los vehículos públicos [14]. Supongamos que un autobús debe pasar a una cierta hora, pero lleva un retraso de 5 minutos, se puede estudiar el caso y

dependiendo a la velocidad que se mueva del punto de parada, avisar al usuario cuando ha de salir de su casa para que no esté esperando al vehículo.

Relacionado con lo anterior, se ha quedado pendiente las áreas que define el propio usuario en la aplicación de Owntracks. Esto se podría explorar en un futuro, pues no sería necesario que el desarrollador defina las áreas sobre las que se van a evaluar la posición del usuario, sino que es el propio usuario el que define estas áreas. Debería estudiarse la forma de añadir en la aplicación que interacción (acercarse, alejarse, entrar, salir…) es la que desea el usuario respecto a un área para así añadirlas al motor de interacciones geolocalizadas y poder llegar a tener una solución más autónoma.

Para concluir, se debería plantear un estudio acerca de la seguridad del protocolo MQTT. Este protocolo se considera el mejor debido a que los mensajes que se traspasan entre dispositivos son ligeros. El problema radica en que, debido a esto, se pueden lanzar una gran cantidad de mensajes simultáneamente, lo que podría ser la causa de un posible ataque DoS.[15]

Bibliografía

[1] Pressman, R. S., & Dr, B. R. M. (2014). Software Engineering: A Practitioner's Approach. McGraw-Hill Education.

[2] Fowler, M. (2018). UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley Professional.

[3] Lauesen, S. (2005). User Interface Design: A Software Engineering Perspective. Pearson Education.

[4] Hohpe, G., & Woolf, B. (2012). Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley.

[5] Fowler, M., & Rice, D. (2003). Patterns of Enterprise Application Architecture. Addison-Wesley Professional.

[6] Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2013). Web Services: Concepts, Architectures and Applications. Springer.

[7] Hwang, H. C., Park, J., & Shon, J. G. (2016). Design and implementation of a reliable message transmission system based on MQTT protocol in IoT. Wireless Personal Communications, 91, 1765-1777.

[8] Challiol, C., Lliteras, A. B., & Gordillo, S. E. (2017). Diseño de aplicaciones móviles basadas en posicionamiento: un framework conceptual. In XXIII Congreso Argentino de Ciencias de la Computación (La Plata, 2017).

[9] Morro Ibáñez, Á. (2021). Creación de una plataforma para la interacción geolocalizada con el mundo físico.

[10] Cadavieco, J. F. (2014, July). Técnicas potentes de interacción con los usuarios de dispositivos móviles: geolocalización y realidad aumentada. In Investigaciones de vanguardia en la universidad de hoy (pp. 221-240). Vision Libros.

[11] Jaraba, F. M. B. (2015). Internet of things. Soluciones pervasivas basadas en geolocalización y near field communication (Doctoral dissertation, Universidad de Córdoba (ESP)).

[12] Orejon-Sanchez, R. D., Crespo-Garcia, D., Andres-Diaz, J. R., & Gago-Calderon, A. (2022). SMART CITIES'DEVELOPMENT IN SPAIN: A COMPARISON OF TECHNICAL AND SOCIAL INDICATORS WITH REFERENCE TO EUROPEAN CITIES. Sustainable Cities and Society, 103828.

[13] P. Jamborsalamati, E. Fernandez, M. J. Hossain and F. H. M. Rafi, "Design and implementation of a cloudbased iot platform for data acquisition and device

supply management in smart buildings", Australasian Universities Power Engineering Conference (AUPEC), pp. 1-6, 2017.

[14] Desai, M., & Phadke, A. (2017, February). Internet of Things based vehicle monitoring system. In 2017 Fourteenth International Conference on Wireless and Optical Communications Networks (WOCN) (pp. 1-3). IEEE.

[15] AP, H. (2019). Secure-MQTT: an efficient fuzzy logic-based approach to detect DoS attack in MQTT protocol for internet of things. EURASIP Journal on Wireless Communications and Networking, 2019(1), 1-15.

ANEXO A

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de desarrollo sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
Ods 1. **Fin de la pobreza**				**X**
Ods 2. **Hambre cero**				**X**
Ods 3. **Salud y bienestar**		**X**		
Ods 4. **Educación de calidad**				**X**
Ods 5. **Igualdad de género**				**X**
Ods 6. **Agua limpia y saneamiento**				**X**
Ods 7. **Energía asequible y no contaminante**				**X**
Ods 8. **Trabajo decente y crecimiento económico**				**X**
Ods 9. **Industria, innovación e infraestructura**		**X**		
Ods 10. **Reducción de las desigualdades**				**X**
Ods 11. **Ciudades y comunidades sostenibles**	**X**			
Ods 12. **Producción y consumo responsable**	**X**			
Ods 13. **Acción por el clima**	**X**			
Ods 14. **Vida submarina**				**X**
Ods 15. **Vida de ecosistemas terrestres**				**X**
Ods 16. **Paz, justicia e instituciones sólidas**			**X**	
Ods 17. **Alianzas para lograr objetivos**				**X**

Reflexión sobre la relación entre el TFG/TFM con los ODS y con el/los ODS

más relacionados.

Como se ha comentado en la fase de análisis sobre todo y se ha apreciado en el resto del TFG, este proyecto pretende automatizar ciertas acciones de la vida cuotidiana para facilitarnos dichas tareas. En los casos de uso se ha demostrado que el framework desarrollado puede llegar a ser una potente herramienta para ciertas acciones o patrones que nosotros mismos hacemos sin pensar a veces, pero pueden llegar a ser una carga. Es por esto por lo que la automatización de ciertos procesos, como apagar las luces de casa cuando no hay nadie, ahorrarnos la tarea de volver a casa si acabamos de salir y cierto electrodoméstico se ha quedado enchufado, son tareas que fácilmente tendrían una solución con el seguimiento de nuestra propia localización.

A continuación, se pretende hacer una relación de los distintos Objetivos de Desarrollo Sostenible que la Agenda 2030 plantea, ya que algunos pueden estar muy ligados a lo que este trabajo de fin de grado aporta como solución.

Por lo referente al **objetivo 11**, este proyecto pretende dar solución a la implantación de un sistema de localización de cada ciudadano de una cierta ciudad, para de esta forma poder avanzar hacía una ciudad totalmente conectada con su entorno tecnológico. Se podría plantear como se ha comentado antes el hecho de tener automatizado las acciones de “salir” o “entrar” en cierta zona, bien sea una casa, un despacho en la oficina, un aula del colegio, para de esta forma enviar un mensaje de “apagar la calefacción” o “encender las luces” y viceversa.

Toda la vida humana actualmente está derivando hacia la innovación. Un ejemplo claro son todos los dispositivos IoT que se tienen instalados en las viviendas. De esto son ejemplos las neveras inteligentes, los Smart thermostat y una larga cantidad de ejemplos. Todo ello se controla desde el dispositivo móvil, por lo que nuestro desarrollo pretende aportar un plus a todo ello, añadiendo la posibilidad de sumar la localización del usuario para así conseguir un gran avance en los **objetivos 9, 12 y 13**.

Otro de los puntos en los que nuestra aplicación puede aportar viene con relación al **objetivo 16**, en el que se aboga por cierta paz en el día a día de las personas. Se debe plantear un caso extremo para entender la relación que guarda este objetivo con nuestro framework. El caso de uso sería una persona que tiene una orden de alejamiento respecto a otra. En este caso, se podría hacer un seguimiento de ambas localizaciones y determinar cuando una cierta persona ha sobrepasado los límites de esa orden, y de esta forma tomar medidas.

Finalmente, todo lo descrito anteriormente contribuye de una forma u otra al **objetivo 3**, buscando mejorar el estilo de vida de las personas.