Escuela Superior
de Ingeniería

ESCUELA SUPERIOR DE INGENIERÍA

GRADO EN INGENIERÍA INFORMÁTICA

SISTEMA EN LA NUBE PARA LA MONITORIZACIÓN Y ALERTA DE LA CALIDAD DEL AIRE EN TIEMPO REAL

AUTOR: DAVID JOSÉ CORRAL PLAZA

Cádiz, julio 2016

Escuela Superior
de Ingeniería

ESCUELA SUPERIOR DE INGENIERÍA

GRADO EN INGENIERÍA INFORMÁTICA

SISTEMA EN LA NUBE PARA LA MONITORIZACIÓN Y ALERTA DE LA CALIDAD DEL AIRE EN TIEMPO REAL

DIRECTORES: JUAN BOUBETA-PUIG, GUADALUPE ORTIZ BELLOT
AUTOR: DAVID JOSÉ CORRAL PLAZA

Cádiz, julio 2016

Agradecimientos

A Juan y Lupe que con su confianza, paciencia y ayuda han hecho posible la realización de este proyecto.

A mis padres Jose Manuel y Cristina, que sin ellos hubiese sido imposible llegar hasta donde estoy hoy.

A mis abuelos Emilio y Lola, que hicieron que los dos primeros años de grado fueran más llevaderos.

A Almudena, que sin su fuerza y apoyo jamás habría logrado este objetivo.

A todos los que me han apoyado y acompañado a lo largo de este viaje...

Mis éxitos son los vuestros.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

En este capítulo se define la temática y motivación del proyecto desarrollado y se describen los objetivos y otros aspectos que guardan relación con este proyecto.

1.1. Motivación

La contaminación atmosférica es un problema que se ha visto agravado en los últimos años debido al creciente número de agentes que se encuentran emitiendo gases contaminantes [41]. Afecta a la salud humana provocando todo tipo de enfermedades respiratorias y cardíacas, así como a la fauna y flora que se encuentran bajo sus efectos. Una evaluación de 2013 realizada por la Centro Internacional de Investigaciones sobre el Cáncer de la Organización Mundial de la Salud (OMS) determinó que la contaminación del aire exterior es carcinógena para el ser humano, y que las partículas del aire contaminado están estrechamente relacionadas con la creciente incidencia del cáncer, especialmente el cáncer de pulmón. También se ha observado una relación entre la contaminación del aire exterior y el aumento del cáncer de vías urinarias y vejiga [9]. Aunque en continentes como Europa se toman constantemente medidas para placar la cantidad de contaminantes que se emiten a la atmósfera, estos problemas persisten y se manifiestan en la calidad del aire. Buena parte de la población europea vive en zonas especialmente urbanas donde la calidad del aire nunca es tan buena como debería de ser.

Actualmente los medios existentes para consultar y consumir información referente a la calidad del aire no disponen de mecanismos que nos permitan consumir dicha información en tiempo real [40], en general es difícil acceder a dicha información ya que se encuentra en ficheros poco legibles cuya finalidad es la automatización y el procesamiento de esos datos más que proporcionar información al usuario de forma tangible. Dicha información se suele encontrar generalizada y, en ningún caso, particularizada para las condiciones y el contexto de un individuo en un momento determinado.

Así pues, la principal motivación de este proyecto es acabar con esta carencia mediante el uso de las tecnologías existentes y de los medios con los que disponemos hoy en día. De este modo, podremos responder a la necesidad de desarrollar un sistema que permita ofrecer un servicio para que el usuario pueda consultar información actualizada y recibir notificaciones adaptada a sus circunstancias tanto personales

(edad, problemas de salud) como contextuales (actividad, ubicación) [71].

Las situaciones detectadas, en el contexto de este proyecto, las identificamos con alertas que notifiquen al usuario de la calidad del aire que se encuentra respirando y de las repercusiones que esto puede conllevar sobre su persona en función de su contexto para poder mejorar la calidad de vida de las personas e intentar concienciarles acerca de este gran problema, que es la calidad del aire. El cálculo de la calidad del aire será llevado a cabo mediante el índice de calidad del aire (*Air Quality Index*, AQI) [1] estadounidense, pudiendo ser extendido a otros índices de calidad del aire existentes en un futuro.

1.2. Objetivos

Los objetivos de este proyecto son los derivados de perseguir una correcta y adecuada monitorización y alerta del nivel de calidad del aire en tiempo real. De este objetivo principal podemos distinguir los siguientes subobjetivos:

- Detectar alertas del nivel de calidad del aire en tiempo real.
- Facilitar al usuario una aplicación móvil con la que pueda ser notificado de cambios en el nivel calidad del aire que le afecten.
- Personalizar, en función del contexto del usuario, la notificación que éste recibirá sobre el evento complejo detectado y notificado.
- Facilitar al usuario un medio para que pueda consultar la calidad del aire de su actual ubicación en tiempo real.
- Almacenar y registrar las alertas de nivel de calidad del aire.

1.3. Alcance

El sistema y la aplicación final pretenden ser destinados a individuos particulares, empresas o instituciones públicas que tengan la necesidad de disponer de un sistema de notificación en tiempo real de cambios en el nivel de la calidad del aire.

Estar respirando un aire de mala calidad tiene graves consecuencias sobre la salud de las personas y más aún cuando estas personas padecen problemas respiratorios, forman parte de algún grupo de la población que es más sensible a la calidad del aire o si simplemente se encuentran realizando algún tipo de esfuerzo en el exterior [33].

La infraestructura necesaria para el despliegue de este proyecto se compone de un servidor que se encargue del procesamiento de los datos y una aplicación móvil

que será ejecutada y utilizada por los diferentes individuos que hagan uso de este sistema.

1.4. Visión general

A continuación realizaremos un recorrido sobre los diferentes capítulos que componen esta memoria. En el capítulo de estado del arte veremos las diferentes tecnologías empleadas en el desarrollo del proyecto, realizando una justificación de su uso. En el capítulo de desarrollo del calendario detallamos las diferentes fases en las que se ha dividido la realización del proyecto e ilustramos dichas fases con un diagrama de Gantt [13].

A continuación, en el capítulo de descripción general del proyecto veremos, de una forma general, diferentes aspectos que guardan relación con el proyecto: requisitos, herramientas empleadas, etc. En el capítulo de implementación, realizaremos un recorrido que comprende desde la fase de diseño hasta la implementación y pruebas realizadas durante el desarrollo del proyecto.

Finalmente, en el capítulo de conclusiones y trabajo futuro, indicamos los resultados que hemos obtenido al término de la realización del proyecto y una serie de mejoras propuestas para una continuación futura del proyecto.

1.5. Glosario

1.5.1. Acrónimos

AQI *Air Quality Index* - Índice de calidad del aire

CEP *Complex Event Processing* - Procesamiento de eventos complejos

EPA *Environmental Protection Agency* - Agencia de protección medioambiental

EPL *Event Processing Language* - Lenguaje de procesamiento de eventos

HTTP *HyperText Transfer Protocol* - Protocolo de transferencia de hipertexto

ICA Índice de calidad del aire

OMS Organización Mundial de la Salud

POJO *Plain Old Java Object*

SSH *Secure SHell* - Intérprete de órdenes seguro

SQL *Structured Query Language* - Lenguaje de consula estructurado

TFG Trabajo de Fin de Grado

UCA Universidad de Cádiz

XML *eXtensible Markup Language*

2. Estado del arte

A continuación describiremos las tecnologías empleadas en el desarrollo del proyecto.

2.1. Procesamiento de eventos complejos

El procesamiento de eventos es un método de seguimiento y análisis de flujos de información (datos) sobre cosas que están ocurriendo (eventos), pudiendo obtener conclusiones sobre ello. El procesamiento de eventos complejos (*Complex Event Processing*, CEP) [38, 32, 75] es el procesamiento de eventos que combina múltiples fuentes de datos para inferir situaciones de interés (patrones de eventos) que sugieren una serie de escenarios o circunstancias más complicadas. El objetivo de CEP es identificar diferentes eventos con diferentes significados y proporcionar una respuesta a ellos lo más acorde y rápido posible. CEP tiene la capacidad de procesar grandes cantidades de eventos y, de estos, quedarse con aquellos que si le proporcionan información valiosa.

Es preciso indicar que al hablar acerca de un motor de reglas y del procesamiento de reglas específicas de negocio, se pueda confundir entre procesamiento de eventos de negocios (*Business Event Processing*, BEP) [8] y CEP; ambos términos corresponden a significados distintos, mientras que el primero se refiere a la orquestación de procesos, el segundo está orientado a la detección de situaciones complejas; no obstante, es posible que CEP, como parte de la respuesta a la detección del evento invoque un *workflow* de BEP. Una de las principales diferencias de CEP con respecto al software de detección de eventos tradicional es su capacidad de detectar eventos en tiempo real, reduciendo así el tiempo de respuesta del usuario ante dicha situación detectada. Así pues, CEP es una tecnología fundamental para aplicaciones que deban responder a situaciones que cambien rápidamente y de forma asíncrona, gestionar excepciones o reaccionar rápidamente a situaciones inusuales; y que requieran adaptabilidad y bajo acoplamiento [54].

Esta tecnología es novedosa e innovadora y se afirma en que necesitaremos, cada vez más, sistemas CEP por varios motivos [70, 72]. Por un lado, los negocios necesitan analizar eventos en tiempo real, debido a que la compañía que analice más rápido su flujo de datos y antes responda a los resultados obtenidos será la más competitiva [68]. Por otro lado, el software tradicional de análisis de eventos no funciona en

tiempo real, por tanto, se pierde tiempo en analizar eventos complejos anteriores, y las bases de datos no están optimizadas para llevar a cabo esta labor.

2.1.1. Esper

Esper [18] es el motor CEP de la compañía EsperTech Inc [18]. Es de código abierto (con licencia GPL [27]) y está disponible en los lenguajes de programación Java y .NET. Tiene la capacidad de procesar y correlacionar miles de eventos por segundo y es capaz de detectar eventos complejos cuando se cumplan una serie de condiciones especificadas previamente en patrones. Estos patrones se escriben siguiendo la sintaxis de Esper, muy parecida al SQL, que se denomina EPL (*Event Processing Language*). Esper dispone de un compilador online (Esper EPL Online [17]) que nos permite definir flujos de eventos, patrones de eventos y realizar la simulación para ver el comportamiento del sistema. Como veremos en los siguientes apartados, uno de los componentes empleados en el desarrollo de este proyecto funciona bajo el motor Esper.

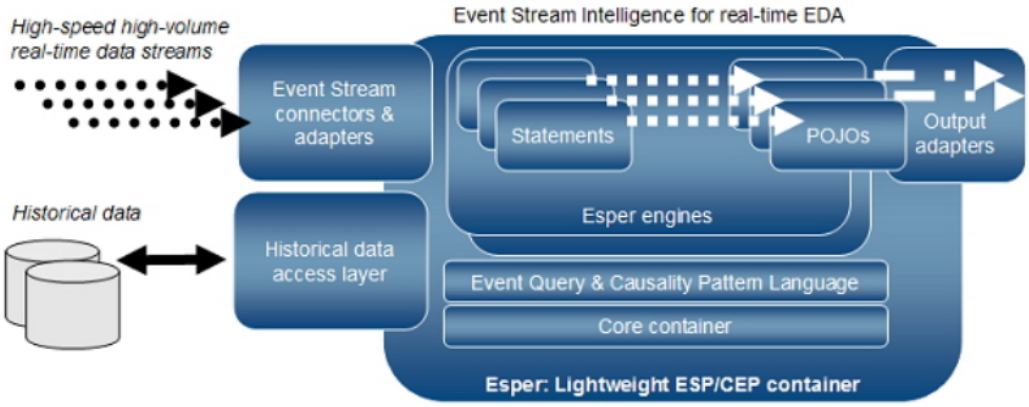

Figura 2.1.: Componentes motor Esper

En la Figura 2.1 podemos observar la arquitectura que muestran los componentes de Esper. Puede observarse que la arquitectura Esper se basa en un contenedor ligero formado por varios componentes. Los tres principales conceptos de la arquitectura Esper son:

- Eventos.
- Patrones de eventos.
- *Listeners*: interfaces para la notificación de eventos.

Este motor acepta varios formatos de eventos como entrada para la detección de eventos —documentos XML, PO-JO (Plain Old Java Object) y *map* de Java—; por lo que Esper es muy adaptable. En Esper, una sentencia en el lenguaje EPL es una consulta que evaluamos continuamente sobre el flujo de eventos que el sistema recibe; estas sentencias pueden ser añadidas en tiempo de ejecución. Los patrones en EPL se registran en el motor Esper y cuando llega un evento, éste se comprueba con cada uno de los patrones que están activos. Cuando uno o más patrones se relacionan con el evento el motor lo notifica a los *listeners* específicos de los patrones de eventos con los que se ha relacionado.

2.1.2. Otros motores CEP

Son muchas las alternativas existentes a Esper en el mercado. Podemos encontrar soluciones tanto de licencia comercial como de licencia libre [74, 73]. A continuación, en la tabla 2.1, recogemos algunas de las soluciones existentes más relevantes.

Tabla 2.1.: Principales motores CEP

Tipo	Nombre	Desarrollador	Ref.
Comercial	Event Zero	Event Zero	[15]
	Microsoft StreamInsight	Microsoft	[43]
	Oracle CEP 10g	Oracle	[48]
	StreamBase	StreamBase Technology	[64]
Libre	Apache Flink	Apache	[5]
	Esper/NEsper	EsperTech Inc.	[18]
	Triceps CEP	Sergey A. Babkin	[67]
	WSO2 CEP	WSO2	[53]

De entre las opciones existentes, Esper se posiciona como una de las mejores del mercado debido a sus múltiples características ya citadas (software libre, gran capacidad de procesamiento, etc.). Esto junto a que el componente empleado para el procesamiento de eventos en el proyecto funciona bajo Esper en su núcleo, hacen que Esper sea nuestra opción principal para el desarrollo de este proyecto.

2.2. Plataforma como servicio

El concepto de Plataforma como Servicio [59] (*Platform as a Service*, PaaS) es una categoría de servicios *cloud* que proporciona una plataforma y un entorno que permiten a los desarrolladores crear aplicaciones y servicios que funcionen a través de internet. Los servicios PaaS se alojan en la nube, y los usuarios pueden acceder a ellos simplemente a través de su navegador web. El modelo PaaS permite

a los desarrolladores crear aplicaciones mediante el uso de diferentes herramientas suministradas por el proveedor. Algunas de las ventajas que ofrece este modelo de desarrollo frente a otros existentes son:

- Escalabilidad, flexibilidad y adaptabilidad.
- Seguridad de los datos y sistemas de recuperación.
- Desarrollo colaborativo.
- No es necesaria una infraestructura física.

Algunas de las grandes empresas que ofrecen estos servicios son Google App Engine [28], que permite el desarrollo de aplicaciones web y móviles altamente escalables desarrollables directamente sobre la estructura de Google, o Amazon Web Services [2], que empezó como un servicio de alquiler de servidores para terminar ofreciendo servicios como IaaS [58] (Infraestructura como Servicio), PaaS o SaaS [60] (Software como servicio).

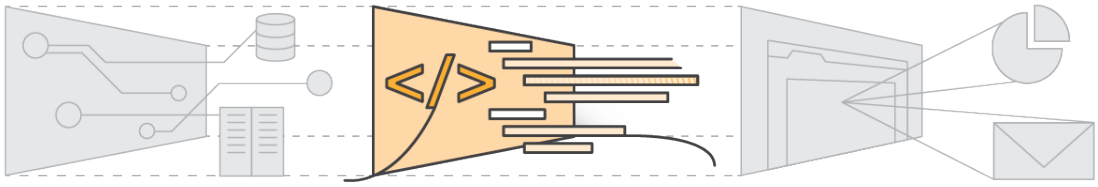

Figura 2.2.: Representación en capas de IaaS, PaaS y SaaS

A pesar de que algunas de las opciones que hemos comentado merecen buena consideración, la herramienta que finalmente acabamos escogiendo para el desarrollo de nuestro proyecto fue FIWARE [24]. FIWARE es una plataforma para el desarrollo y despliegue de aplicaciones en la nube [10], enfocada al perfil de aplicaciones de Internet de las cosas [37] (*Internet of the Things*, IoT). FIWARE fue iniciado como un co-proyecto impulsado por la Comunidad Económica Europea y grandes compañías como IBM, Telefónica (ahora Movistar), Orange, etc. A día de hoy, 24 de abril de 2016, dispone de casi 3000 núcleos de procesamiento, 10000 GB de memoria RAM, 566 TB de almacenamiento en discos duros, repartidos por 12 países, y cerca de 5500 direcciones de IP públicas. De dicho hardware, alrededor del 50 % se encuentra ubicado en el nodo de España, en Madrid.

FIWARE funciona mediante la interconexión de diferentes componentes que desempeñan funcionalidades específicas y, a su vez, se complementan unos con otros para que, mediante la unión de los componentes adecuados, podemos llegar a desarrollar la aplicación que perseguida. Estos componentes desempeñan funciones tales como

procesamiento de eventos complejos, gestión de contextos, análisis de Big Data [57], etc. Los dos componentes utilizados en el desarrollo de este proyecto han sido *IoT Data Edge Consolidation GE* - Cepheus [21] y *Publish/Subscribe Context Broker* - Orion Context Broker [23].

2.2.1. Cepheus

El componente Cepheus es un motor CEP que además dispone de otras funcionalidades añadidas y está basado en Esper. Cepheus se divide a su vez en el Cepheus CEP y el Cepheus Broker, siendo el primero el encargado del procesamiento de eventos complejos y el segundo el encargado de la comunicación del motor CEP con el resto de componentes del sistema, principalmente. Además Cepheus Broker dispone de una API REST [11] la cual permite operaciones tales como *GET*, *POST* o *DELETE* [31]. El paso de mensajes y sintaxis funciona con el modelo de datos NGSI 9/10 [20], una interfaz desarrollada por FIWARE para el manejo de datos entre sus componentes.

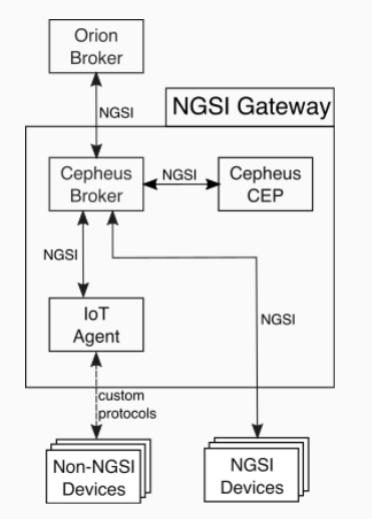

Figura 2.3.: Esquema Cepheus CEP

2.2.2. Orion Context Broker

El componente Orion Context Broker es un gestor de contextos que permite gestionar todo el ciclo de vida de la información de contexto incluyendo actualizaciones, consultas, registros y suscripciones. Utilizando Orion Context Broker podremos registrar los elementos de contexto y administrarlos a través de actualizaciones y consultas. Además, permite la suscripción a la información de contexto de modo que

cuando se da alguna condición determinada (por ejemplo, un intervalo de tiempo ha pasado o los elementos del contexto han cambiado) recibe una notificación. Es el complemento por excelencia del Cepheus CEP y sirve de paso intermedio entre éste y otros componentes como Cosmos [22], el componente de Big Data de FIWARE.

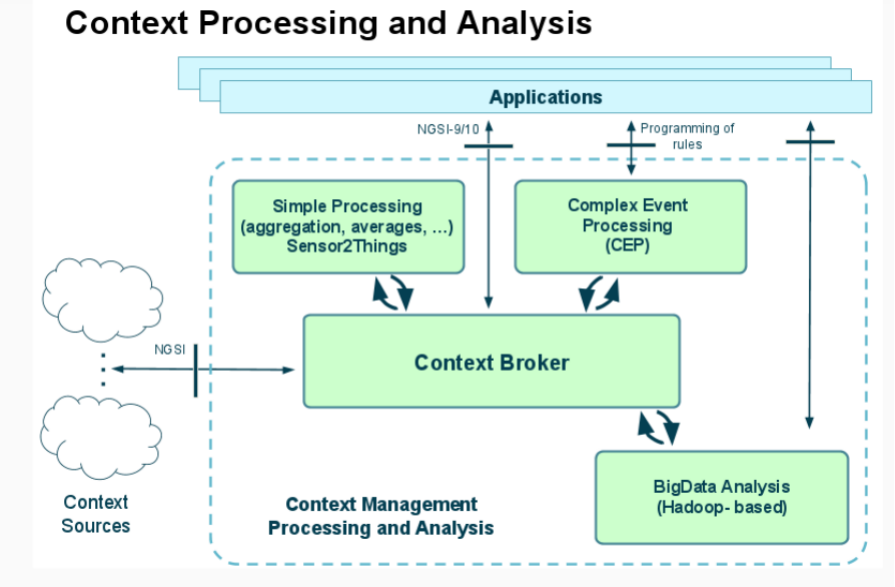

Figura 2.4.: Esquema Orion Context Broker

2.3. Tendencias en dispositivos inteligentes

Actualmente la tendencia de tenerlo todo en un mismo dispositivo se ha visto disparada gracias a la aparición de los dispositivos inteligentes. Estos dispositivos suelen funcionar con complejos sistemas operativos que permiten la instalación de diferentes aplicaciones.

Para la realización de este proyecto hemos elegido Android [3], que es el sistema operativo de Google (creado por Android Inc), basado en Linux y originalmente orientado a dispositivos móviles inteligentes (*smartphones*). Se ha ido expandiendo a otros dispositivos de uso cotidiano como tabletas (*tablets*), tabléfonos (*phablets*) y también a dispositivos cuya evolución hacia la inteligencia se ha visto disparada en los últimos años como son las televisiones inteligentes (*Smart-TV*) y los dispositivos portables (*wereables*) tales como relojes inteligentes, pulseras inteligentes, etc. Esta plataforma es de código abierto y permite el desarrollo de aplicaciones por terceros (personas ajenas a Google). La mayoría del código fuente de Android ha sido publicado bajo la licencia de software Apache [42], una licencia de software libre y código

fuente abierto.

Aunque esté escrito principalmente en los lenguajes de programación C y C++, a la hora de desarrollar el código que se encargará de llevar a cabo la mayoría de las funcionalidades de la aplicación las hemos desarollado en el lenguaje de programación Java. El almacenamiento de datos lo realizamos haciendo uso de bases de datos SQL, concretamente SQLite [52]. El uso de estas tecnologías hacen que Android sea la opción número uno para los desarrolladores de todo el mundo a la hora de programar sus aplicaciones frente a otros sistemas operativos como iOS [39], el sistema operativo de Apple que ocupa la segunda plaza en cuanto a número de dispositivos con su sistema operativo instalado [26]. Aunque su rendimiento, en algunos aspectos, es muy superior a Android, su perfil de sistema operativo cerrado, la necesidad de un equipo especial para llevar a cabo el desarrollo de aplicaciones iOS y la exclusividad que muchas veces implican algunos de sus productos, son algunas de los enemigos (y también aliados en otros aspectos) de este sistema operativo a la hora de ser escogido como opción de desarrollo.

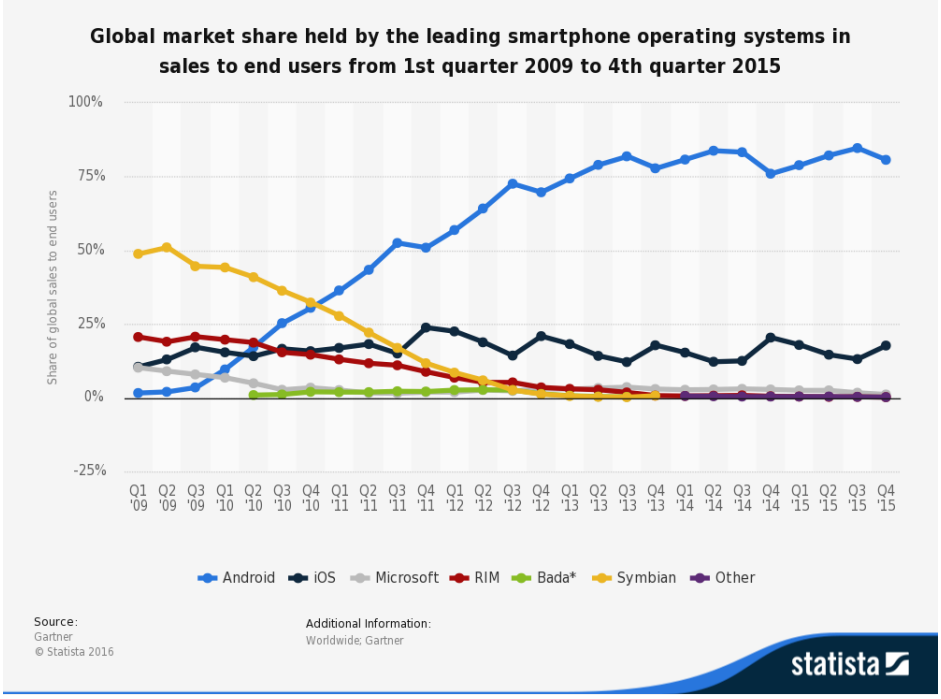

Figura 2.5.: Tasa de mercado en 2015 de SO en *smartphones*

2.4. Calidad del aire

La contaminación del aire representa un importante riesgo medioambiental para la salud de las personas y del resto de fauna y flora existente en el mundo. Un requisito básico de una buena salud y un bienestar humano es una buena calidad del aire. La calidad del aire afecta a cómo vives y respiras; puede cambiar de un día para otro o incluso de una hora para otra. Cada vez son más los elementos que encontramos emitiendo gases contaminantes a la atmósfera y, por lo tanto, mayores son los riesgos a los que nos encontramos expuestos, sobre todo en grandes áreas urbanas en las cuales el nivel de calidad del aire nunca es lo suficientemente bueno. Según las últimas estimaciones de la OMS [47] sobre la carga mundial de morbilidad, la contaminación del aire exterior e interior provoca unos siete millones de defunciones prematuras [9]. Esto representa actualmente uno de los mayores riesgos sanitarios mundiales, comparable a los riesgos relacionados con el tabaco, y superado únicamente por los riesgos sanitarios relacionados con la hipertensión y la nutrición.

Existen numerosos ejemplos de políticas fructíferas relativas a los sectores de transporte, planificación urbana, generación de electricidad e industria, que permiten reducir la contaminación del aire:

- Industria: utilización de nuevas tecnologías ecológicas que permitan reducir las emisiones que las chimeneas industriales produce.
- Transporte: adopción de nuevas tecnologías como coches con motores híbridos o utilización de medios de transporte público.
- Generación de electricidad: hacer uso de energías renovables como solar o eólica.

La calidad del aire se evalúa en función de los valores que toman ciertos contaminantes. Existen muchos índices para evaluar los contaminantes y obtener un nivel de calidad del aire:

- AQI - *Air Quality Index* [1]: Es el índice de calidad del aire diario acuñado por la agencia de protección medioambiental *(Environmental Protection Agency,* EPA) [16] de Estados Unidos. Se basa en seis contaminantes: Ozono, Dióxido de Nitrógeno, Dióxido de Azufre, Partículas en Suspensión de hasta 10 micrometros de diámetro, Partículas en Suspensión de hasta 2.5 micrometros de diámetro, Monóxido de Carbono.
- AQI - *The National Air Quality Index* [44]: Es el índice de calidad del aire acuñado por el Ministerio de Medio ambiente, Bosques y Cambio climático en la India. Se basa en ocho contaminantes: Ozono, Dióxido de Nitrógeno, Dióxido de Azufre, Partículas en Suspensión de hasta 10 micrometros de diámetro,

Partículas en Suspensión de hasta 2.5 micrometros de diámetro, Monóxido de Carbono, Amoníaco y Plomo.

- ICA - Índice de Calidad del Aire [45]: es el índice de calidad del aire utilizado en España. Se basa en cinco contaminantes: Ozono, Dióxido de Nitrógeno, Dióxido de Azufre, Partículas en Suspensión de hasta 10 micrometros de diámetro, Monóxido de Carbono.

2.4.1. AQI

Para la implementación de los patrones de eventos de nuestro proyecto hemos decidido emplear el AQI, que es el sistema de medición para la calidad del aire acuñado y empleado por la EPA, que se encarga de realizar reportes diarios sobre la calidad del aire y se centran, principalmente, en cómo la calidad del aire puede afectar a nuestra salud tras respirar durante un tiempo determinado aire de mala calidad.

Actualmente presenta una estandarización para los valores de los contaminantes Ozono, Dióxido de Azufre, Partículas en Suspensión de hasta 10 micrometros de diámetro, Partículas en Suspensión de hasta 2.5 micrometros de diámetro, Monóxido de Carbono. En próximas revisiones está prevista una estandarización para el contaminante Dióxido de Nitrógeno.

De igual forma, el índice de calidad del aire se mide con valores comprendidos entre 0 y 500. A cada nivel AQI se le asigna un color y una condición como podemos observar en la figura 2.6:

Para cada uno de estos contaminantes citados existen un rango de valores determinados, figura 2.7, y una serie de recomendaciones o advertencias asociadas a dichos valores para que, en función de cual es la calidad del aire, las personas que se encuentren afectadas por dicha calidad realicen determinadas acciones. Por ejemplo, en la figura 2.8 podemos apreciar las diferentes recomendaciones a llevar a cabo para diferentes valores AQI del contaminante Monóxido de Carbobo.

Air Quality Index (AQI) Values	Levels of Health Concern	Colors
When the AQI is in this range:	*...air quality conditions are:*	*...as symbolized by this color:*
0 to 50	Good	Green
51 to 100	Moderate	Yellow
101 to 150	Unhealthy for Sensitive Groups	Orange
151 to 200	Unhealthy	Red
201 to 300	Very Unhealthy	Purple
301 to 500	Hazardous	Maroon

Figura 2.6.: Tabla valores AQI

POLLUTANTS						AQI	
O3 (ppm) 8-hour avg	CO (ppm) 8-hour avg	PM10 (µg/m³) 24-hour avg	PM2,5 (µg/m³) 24-hour avg	SO2 (ppm) 1-hour avg	NO2 (ppm) 1-hour avg	AQI	Category
0.000 – 0.064	0.0 – 4.4	0 – 54	0.0 – 15.4	0.000 – 0.034	----	0 – 50	Good
0.065 – 0.084	4.5 – 9.4	55 – 154	15.5 – 40.4	0.035 – 0.144	----	51 – 100	Moderate
0.085 – 0104	9.5 – 12.4	155 – 254	40.5 – 65.4	0.145 – 0.224	----	101 – 150	Unhealthy for Sensitive Groups
0.105 – 0.124	12.5 – 15.4	255 – 354	65.5 – 150.4	0.225 – 0.304	----	151 – 200	Unhealthy
0.125 – 0.374	15.5 – 30.4	355 – 424	150.5 – 250.4	0.305 – 0.604	0.65 – 1.24	201 – 300	Very unhealthy
>= 0.375	>= 30.5	>= 425	>= 250.5	>= 0.605	>= 1.25	>= 300	Hazardous

Figura 2.7.: Valores contaminantes AQI

AQI Value	Actions To Protect Your Health From Carbon Monoxide
Good (0–50)	None
Moderate (51–100*)	None
Unhealthy for Sensitive Groups (101–150)	People with heart disease, such as angina, should reduce heavy exertion and avoid sources of carbon monoxide, such as heavy traffic.
Unhealthy (151–200)	People with heart disease, such as angina, should reduce moderate exertion and avoid sources of carbon monoxide, such as heavy traffic.
Very Unhealthy (201–300)	People with heart disease, such as angina, should avoid exertion and sources of carbon monoxide, such as heavy traffic.

Figura 2.8.: Tabla recomendaciones AQI CO

3. Desarrollo del calendario

A continuación definiremos las fases en las cuales hemos desarrollado el proyecto y un diagrama de Gantt que ilustre dicho procedimiento de forma gráfica.

3.1. Fases

En esta sección detallaremos el trabajo llevado a cabo durante cada una de las fases del proyecto.

3.1.1. Fase primera: Inicio del proyecto

El proyecto comienza con el descubrimiento de la plataforma FIWARE. Tras una serie de reuniones con los tutores, acordamos que esta será la plataforma sobre la cual desarrollaremos el proyecto aprovechando sus múltiples ventajas. Barajamos diferentes temáticas para el trabajo pero finalmente nos decantamos por la calidad del aire. Por lo tanto, definimos una serie de primeros objetivos entre los cuales estaba la detección en tiempo real de la calidad del aire por localización en función de la información que recibimos desde diferentes sensores. Aunque algunos de estos primeros objetivos, en aquel momento, eran simplemente un punto de referencia para ir estableciendo plazos, más adelante nos dimos cuenta de que pasarían a ser elementos más de nuestro proyecto.

3.1.2. Fase segunda: Trabajo de investigación

Al ser FIWARE una herramienta que, prácticamente, acaba de surgir, el trabajo de investigación es una obligación ya que tanto la documentación oficial como la disponible en línea es escasa. Es el momento de ponerse delante de dicha plataforma, aprender a manejarla e ir logrando, en la medida de lo posible, entender y comprender sus diferentes funcionalidades que más tarde nos conducirían al desarrollo de nuestro proyecto. Por lo tanto, durante esta fase, recopilamos información y realizamos algunos proyectos de ejemplo para probar y descubrir el funcionamiento en líneas generales de esta plataforma. Durante esta fase contábamos con una cuenta que disponía de un período de prueba de unos 30 días de validez. Cuando el período de prueba iba a expirar, solicitamos una cuenta de la comunidad comentando los

avances que habíamos logrado y el camino que estaba dibujando nuestro proyecto. La petición fue aceptada. Durante esta fase llegamos a la conclusión de cuáles serían los componentes de FIWARE que tendríamos que emplear para el desarrollo de nuestro sistema de detección de eventos, dichos componentes son IoT Data Edge Consolidation GE - Cepheus y Publish/Subscribe Context Broker - Orion Context Broker. Desde los comienzos de esta fase pusimos en marcha la Redacción de la Memoria la cual la hemos desarrollado durante las diferentes fases para ir documentándolas.

3.1.3. Fase tercera: Detección de eventos con Cepheus

Una vez que hemos comprendido y asimilado los conceptos y el funcionamiento de la plataforma FIWARE e identificado los componentes a emplear, era el momento de ponerse a trabajar con ellos. En primer lugar desarrollamos un pequeño sistema de detección de eventos usando el componente Cepheus, el cual simplemente recibía los datos mediante un script que ejecutábamos en la terminal del sistema, la configuración con los patrones de eventos y, cuando detectábamos un evento complejo por parte del motor CEP, mostrábamos por pantalla el evento complejo detectado y los valores de sus atributos. Para la realización de estas pequeñas pruebas realizamos los tutoriales disponibles en la documentación oficial del componente.

3.1.4. Fase cuarta: Gestión de contextos y suscripción con Orion

Por otra parte, desarrollamos un pequeño proyecto con Orion Context Broker en el cual enviábamos información, la actualizábamos y la recuperábamos para comprender su estructura, el funcionamiento de su API REST y demás características de Orion. Realizamos algunas pruebas con la característica de suscripción de Orion que nos permitió comprobar el funcionamiento de esta funcionalidad que desempeñaría un papel fundamental en la arquitectura de nuestro proyecto. Una vez más, para la realización de estas pequeñas pruebas realizamos los tutoriales disponibles en la documentación oficial del componente.

3.1.5. Fase quinta: Fusión de Cepheus y Orion

Con el dominio y conocimiento necesarios para manipular y trabajar estos dos componentes, era el momento de unificarlos y hacerlos trabajar de forma conjunta. Para ello creamos una máquina virtual en la plataforma FIWARE en la cual instalamos y configuramos Orion Context Broker y Cepheus, en dicho orden.

Una vez comprobado el funcionamiento de ambos componentes, por separado, en una misma máquina, pasamos a la interconexión de dichos componentes para montar el esqueleto principal de lo que sería nuestro proyecto. En este esquema,

el Broker de Cepheus recibiría la información (eventos simples) y se encargaría de enviarlos al motor Cepheus CEP para su procesamiento y al gestor de contextos Orion Context Broker para que almacene la información correspondiente. En el momento que detectásemos un evento complejo, el motor CEP notifica al Broker y este hace lo propio notificando a Orion. Durante esta fase desarrollamos lo que sería el esqueleto principal del sistema de detección de eventos complejos de nuestro proyecto de una forma íntegra.

3.1.6. Fase sexta: Suscripción y registro de alertas

Una vez tenemos el esqueleto de la aplicación en funcionamiento y detectando eventos complejos, es el momento de llevar a cabo la suscripción a dichos eventos complejos. Para dicha finalidad agregamos un módulo escrito en el lenguaje de programación Python que, mediante un comando determinado, se suscribe a los eventos complejos de un tipo determinado al Orion bajo ciertas condiciones. Cuando se dan dichas condiciones, Orion notifica al módulo Pyhton que se encarga de almacenar las alertas en un registro de una base de datos local, que funciona bajo el servidor LAMPP que es un servidor independiente de plataforma, software libre, que consiste principalmente en el sistema de gestión de bases de datos MySQL [7], el servidor web Apache y los intérpretes para lenguajes como PHP. Al finalizar esta fase, prácticamente todo el sistema de detección y suscripción de eventos complejos está planteado y finalizado.

3.1.7. Fase séptima: Definición de patrones de eventos

Con la estructura de la detección de eventos completada y en funcionamiento, era el momento de definir los diferentes patrones de eventos que nuestro sistema iba a ser capaz de detectar a partir de los datos de entrada. Para esta fase implementamos los diferentes patrones de calidad del aire según el estándar estadouninendese, el AQI. En primer lugar implementamos estos patrones en EPL y probamos su correcto funcionamiento en el compilador online que Esper tiene para tal fin.

3.1.8. Fase octava: Prueba del sistema de detección de eventos

Ya con los patrones definidos y con la estructura de detección de eventos desarrollada, era el momento de poner en marcha todo de forma conjunta y forzar al sistema a que fuese capaz de detectar diferentes eventos complejos definidos previamente. Llevamos a cabo una simulación de datos que forzasen la ocurrencia de los eventos complejos ya que, debido a la naturaleza del trabajo (calidad del aire), usar datos reales y esperar a que se diesen las condiciones para la detección de dichos eventos es inviable. En esta fase no solo probamos la detección de eventos complejos, sino

también su suscripción y posterior notificación al módulo encargado de ello. Durante esta fase llevamos a cabo la resolución de errores que surgieron de la propia integración de los componentes.

3.1.9. Fase novena: Desarollo Android

Dejando de lado la detección de eventos complejos en FIWARE, es necesario proporcionar una aplicación con la cual el usuario pueda ser notificado de los diferentes eventos complejos detectados y, más importante aún, que tenga en cuenta cual es su contexto actual para ser notificado consecuentemente. Por contexto entendemos la actividad que se encuentra realizando el usuario y la localización del mismo en un instante determinado. Para ello, hacemos uso del sensor GPS [69] del que disponen los teléfonos inteligentes para obtener las coordenadas y la velocidad de movimiento. Con estos dos datos será posible determinar, siempre de forma aproximada, dónde se encuentra el usuario y que acciones está realizando. Por ejemplo:

- Si la velocidad es inferior a 1 metro por segundo, el usuario se encuentra estático.
- Si la velocidad está comprendida entre 1 metro por segundo y 2.7 metros por segundo, el usuario se encuentra caminando.
- Si la velocidad está comprendida entre 2.7 metros por segundo y 5.5 metros por segundo, el usuario se encuentra realizando alguna actividad de esfuerzo, como correr.
- Si la velocidad es superior a 5.5 metros por segundo, el usuario irá en algún medio de transporte, coche, autobús, etc.

Por otro lado, desarollamos un modo alternativo al uso del sensor GPS que permite al usuario introducir una serie de hábitos para su día a día de tal forma que él, para determinados periodos de tiempo, define cual será la actividad que estará realizando y la ubicación, es decir, dónde se encontrará realizándolas. De esta forma evita un gasto considerable de batería ya que el sensor GPS consume muchos recursos.

3.1.10. Fase décima: Notificaciones en tiempo real

Para las notificaciones en tiempo real hemos hecho uso de un *framework [61]* que permite emplear Google Cloud Messaging [29] sin tener la necesidad de desarrollar la parte del servidor. Dicho *framework* o librería es PushBots [56] que ofrece un sistema de notificaciones en tiempo real, basado en Google Cloud Messaging y con algunas funcionalidades extras como la asignación de TAG's. Un TAG es una cadena

que se le asigna a un dispositivo registrado en el sistema previamente y que permite identificarlo con un grupo determinado. Por lo tanto, en función del contexto y de la localización, asignamos TAG's específicos para cada uno de los dispositivos registrados en el sistema y, en el momento que detectemos un evento complejo que afecte a una localización determinada, los dispositivos que han sido registrados con el TAG asignado a dicha localización serán notificados.

Las notificaciones las creamos en diferentes Servicios Webs que, en función del evento complejo a notificar, reciben unos parámetros y con ellos notifican a los usuarios afectados por dicho evento complejo detectado. El Servicio Web es invocado por el consumidor de eventos que se suscribe a los eventos complejos del tipo que deseamos ser notificado.

3.1.11. Fase undécima: Pruebas y depuración

Finalmente, con todo el sistema desarrollado, hemos llevado a cabo una serie de pruebas en todas y cada una de las partes que componen este proyecto, desde la detección de eventos hasta el correcto funcionamiento de las diferentes funcionalidades desarrolladas en la aplicación Android. Una vez realizadas estas pruebas unitarias, hemos realizado pruebas de integración con el objetivo de verificar el correcto funcionamiento del sistema de forma conjunta. En esta fase se han corregido los últimos errores que hemos detectado durante la realización de pruebas y ejecución del sistema. Dichas pruebas las describimos en el capítulo cinco de esta memoria.

3.2. Diagrama de Gantt

A continuación incluimos el Diagrama de Gantt en el que podemos visualizar el desarrollo de las diferentes fases de forma gráfica. El diagrama ha sido desarrollado con la herramienta GanttProject [25].

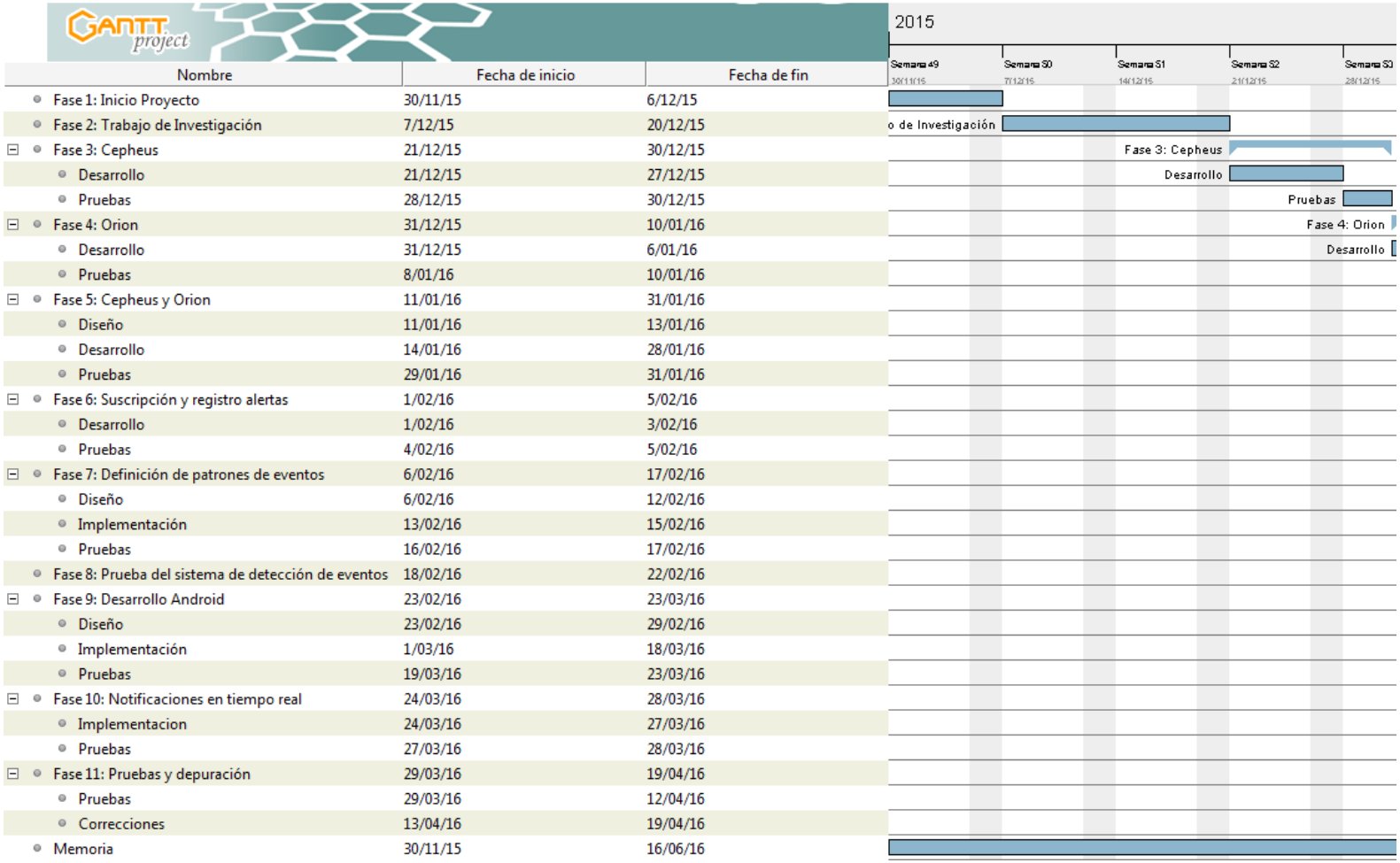

Figura 3.1.: Diagrama de Gantt

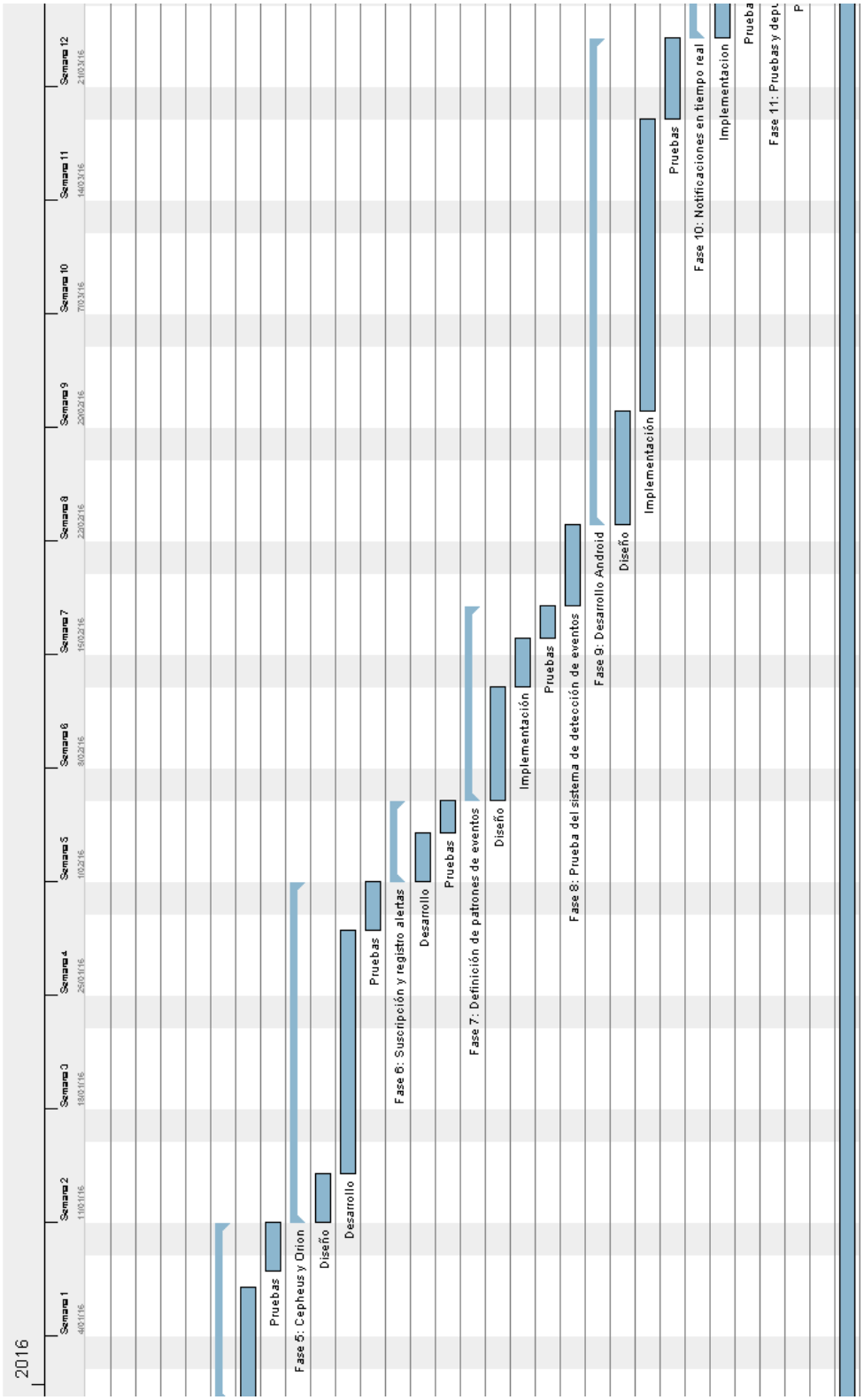

Figura 3.2.: Diagrama de Gantt

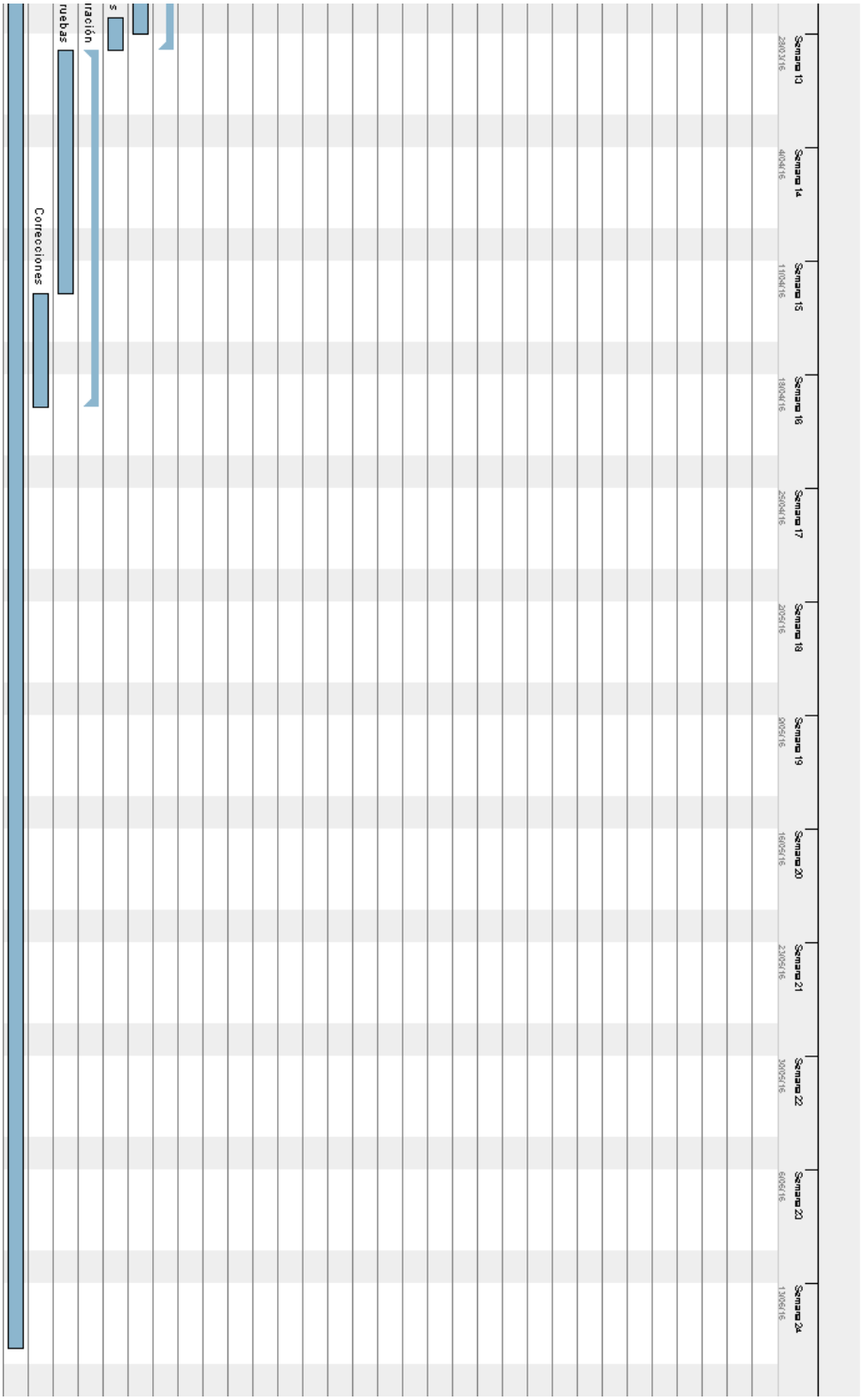

Figura 3.3.: Diagrama de Gantt

4. Descripción general del proyecto

En este capítulo detallaremos las diferentes funcionalidades y características del proyecto.

4.1. Perspectiva del producto

4.1.1. Interfaz de usuario

En nuestro proyecto desarrollado, la interfaz de usuario como tal se identificaría con la aplicación Android que el usuario final tendrá instalada en su dispositivo o teléfono inteligente. La aplicación ha sido desarrollada con el software Android Studio [4] permitiendo ser testada y previsualizada en diferentes tamaños de pantalla, asegurando una correcta visualización en dispositivos con pantallas de tamaño comprendido entre 3.5 pulgadas y 11 pulgadas. En esta aplicación el usuario tendrá disponibles las diferentes funcionalidades que describiremos en el siguiente apartado.

Para la parte de detección de eventos no existe una interfaz gráfica ya que todos los componentes de FIWARE los ejecutamos en terminales del sistema.

4.2. Funciones

A continuación listamos las funcionalidades del proyecto, incluyendo tanto las propias del sistema de procesamiento de eventos como las de la aplicación final Android.

- Detección de eventos complejos relativos a la calidad del aire en tiempo real.
- Notificación por uso de Google Cloud Messaging de alertas del nivel de la calidad del aire.
- Personalización de notificación en función del contexto, la edad y posibles enfermedades respiratorias.
- Posibilidad de realizar una monitorización constante del contexto mediante el uso del sensor GPS del dispositivo o bien introducir en un horario el tipo de actividad que el usuario estará desempeñando en cada momento determinado del día.

- Consultar el estado de la calidad del aire de tu ubicación en tiempo real.
- Registro y almacenamiento de alertas detectadas en una base de datos MySQL.

4.3. Características del usuario

El usuario final que hará uso de esta aplicación no tiene un perfil concreto puesto que respirar es una acción que toda persona humana lleva a cabo y estar al corriente de la calidad del aire que respiramos y de las consecuencias que puede acarrear para nuestra salud son aspectos interesantes para cualquier usuario. Si tuviéramos que definir un público ideal al que destinar este proyecto, serían personas que padezcan algún tipo de enfermedad respiratoria o cardíaca, personas mayores, menores de edad y demás grupos sensibles de la población a los cuales la calidad del aire les afecta de una manera más crítica que al resto.

Por otra parte, el sistema ha de ser gestionado y administrado por usuarios con conocimientos relativos a la temática que trabajamos, la calidad del aire, y con conocimientos avanzados de informática, computación en la nube y procesamiento de eventos complejos.

4.4. Lenguajes de programación y tecnologías

Durante las fases del proyecto hemos empleado diferentes lenguajes de programación y tecnologías.

- **FIWARE:** como tecnología principal y núcleo de la carga de procesamiento de la información y notificación de los eventos complejos detectados a los usuarios. Es una plataforma de computación en la nube que ofrece la creación de máquinas virtuales con hardware personalizable. Dentro de dichas máquinas podemos instalar diferentes componentes que desempeñaran ciertas funcionalidades.
- **CEP:** utilizado para el procesamiento de eventos complejos. Para dicha tarea hemos empleado el componente Cepheus de FIWARE que funciona con el motor y sintaxis de Esper en el núcleo de procesamiento.
- **Gestión de contextos:** hemos empleado el componente Orion Context Broker de FIWARE para la gestión de contextos y la suscripción a los diferentes cambios en los eventos complejos detectados por el motor CEP.
- **Servidor Apache:** con soporte de base de datos MySQL (phpMyAdmin [50]) y ejecución de scripts en PHP que han hecho posibles el registro de las alertas

detectadas en la base de datos y despliegue de los Servicios Web empleados para la notificación y comunicación con la aplicación Android.

- **Android:** como sistema operativo para el desarrollo de la aplicación final con la cual el usuario hará uso de las diferentes funcionalidades de este proyecto.
- **Lenguajes de programación:**
  - **PHP:** para los Servicios Web que comunican al sistema de procesamiento con la aplicación Android.
  - **Java:** para el desarrollo de las diferentes funcionalidades de la aplicación Android.
  - **Python:** para el desarrollo del módulo que se encarga de invocar a un Servicio Web determinado cuando se detecta un evento complejo.

4.5. Herramientas

Durante las fases del proyecto hemos empleado diferentes herramientas.

- **Android Studio:** ha sido la herramienta principal en el desarrollo de la aplicación Android. Android Studio es un entorno de desarrollo para aplicaciones Android. Está basado en el software IntelliJ IDEA [36] de JetBrains, y es publicado de forma gratuita a través de la Licencia Apache 2.0. Disponible para Windows, GNU/Linux y Mac OS X.
- **Sublime Text** [65]: ha sido empleado la mayoría del tiempo para la escritura y modificación de los diferentes ficheros de configuración del sistema desplegado en la plataforma FIWARE. Es un editor de texto y editor de código fuente está escrito en C++ y Python para los plugins. Aunque su licencia es de Software propietario, dispone de una versión gratuita, funcional y sin fecha de caducidad.
- **Terminator** [66]: es una aplicación para la visualización de múltiples terminales en una sola ventana. Debido a que la escritura, modificación y lectura de ficheros de configuración del sistema la hacemos vía SSH [55], es necesario disponer de un software que permita una buena división del monitor en diferentes terminales que aseguren un trabajo eficiente en la fuente destino.

4.6. Sistemas operativos y hardware

Durante la mayor parte del desarrollo del proyecto hemos empleado un portátil Acer V3-571G con el sistema operativo Ubuntu 14.04. En dicho portátil hemos

desarrollado la aplicación Android y hemos llevado a cabo las conexiones vía SSH al servidor en la nube dónde desplegamos nuestro proyecto. Sin embargo, el proyecto, como hemos comentado anteriormente, lo desplegamos en un servidor en la nube cuyo hardware es 4GB de memoria RAM, 2 unidades de procesamiento y 40GB de disco duro, además de una IP pública y otra privada. El sistema operativo de dicho servidor es CentOS.

4.7. Control de versiones

Debido a que la mayor parte del trabajo se ha realizado vía SSH sobre los diferentes ficheros de configuración en la plataforma FIWARE, hemos ido realizando copias de seguridad de forma progresiva de estos ficheros en un repositorio Git de la plataforma Neptuno [46] que pertenece al grupo de investigación UCASE [30]. Para el desarrollo de la aplicación Android hemos llevado a cabo las mismas acciones, realizando copias de seguridad del código fuente de la aplicación en el mismo repositorio.

5. Desarrollo del proyecto

En este capítulo explicaremos los aspectos más importantes y técnicos del proyecto desarrollado, incluyendo secciones de código fuente de los diferentes sistemas siempre que sea necesario para una mejor comprensión.

5.1. Modelo de ciclo de vida

Desde un primer momento intuimos que definir, de forma inequívoca, las funcionalidades y características del proyecto desde un primer momento era una tarea prácticamente imposible ya que, debido a la propia naturaleza del área que trabajamos y a las herramientas que decidimos emplear, las diferentes funcionalidades de este proyecto se irían descubriendo, definiendo y añadiendo sobre la marcha. Por lo tanto, y teniendo en cuesta esto, el modelo de ciclo de vida de este proyecto queda identificado con el ciclo de vida incremental.

Este tipo de modelo de ciclo de vida surge como respuesta a las debilidades que presenta el modelo en cascada. Es un modelo que agrupa diferentes etapas repetitivas que permite al desarrollado sacar provecho de lo que hasta el momento lleva implementado. La principal premisa de este modelo es que, durante el desarrollo, los procesos o necesidades tienden a cambiar, esto es pues la realidad de este proyecto debido a que en función que avanzábamos por las diferentes fases descritas anteriormente, podíamos ampliar el funcionamiento del sistema con nuevas características descubiertas.

5.2. Requisitos

A continuación detallaremos los requisitos del proyecto de forma completa.

5.2.1. Funcionales

Los requisitos funcionales que debe cumplir el sistema son:

- **Procesamiento de eventos simples.** El sistema debe de ser capaz de recibir datos (eventos simples) de una fuente de información, almacenarlos y procesarlos.

- **Detección de eventos complejos.** El sistema debe de ser capaz de, a partir de los eventos simples de entrada y la definición de unos patrones de eventos, generar información relevante y de interés, que se traduce en detectar eventos complejos en tiempo real.
- **Notificación de eventos complejos.** El sistema debe de ser capaz de notificar en tiempo real de los eventos complejos detectados al consumidor correspondiente, previamente suscrito, que se encargará de notificar al usuario.
- **Monitorización del contexto.** El sistema debe de ser capaz de monitorizar el contexto del usuario con el fin de poder asignarle un TAG en el sistema de notificaciones acorde a su contexto. Esta monitorización puede ser mediante el uso del GPS o bien mediante el horario prestablecido por el usuario.
- **Personalización de notificación.** El sistema debe de ser capaz de notificar de una forma personalizada y adecuada al contexto del usuario que recibe la notificación, siendo por lo tanto la notificación diferente en función del tipo de evento y usuario a notificar.
- **Registro de eventos complejos.** El sistema debe de ser capaz de registrar los diferentes eventos complejos detectados en una base de datos.
- **Datos y preferencias del usuario.** El sistema debe de ser capaz de ofrecer al usuario la posibilidad de suscribirse o no a las alertas de determinados eventos complejos y, además, darle la opción de que indique su rango de edad y si padece o no alguna enfermedad respiratoria.
- **Consultar alertas de calidad.** El sistema debe de ser capaz de ofrecer al usuario la posiblidad de consultar las últimas alertas de calidad del aire detectadas para su ubicación.

5.2.2. De información

Los requisitos de información que debe cumplir el sistema son:

- El sistema almacenará los datos de los eventos complejos detectados en una base de datos.
- El sistema proporcionará una serie de recomendaciones en función del usuario y el evento complejo notificado.
- El sistema sabrá, en todo momento, cual es el actual (o último) TAG del usuario.

5.2.3. Interfaz

El sistema debe de ser capaz de conectarse con todos los elementos que lo componen. Estos son: la plataforma FIWARE, dónde se lleva a cabo el procesamiento de eventos complejos, la gestión de contextos y la notificación a usuario, los diferentes Servicios Web que proveen una comunicación entre FIWARE y la aplicación Android, que será con la cual el usuario final interactúe y el medio que tendrá este de acceder a las diferentes características y funcionalidades.

5.2.4. No funcionales

Los requisitos no funcionales del sistema son:

- **Tiempo de respuesta.** El sistema debe de ser capaz de funcionar en tiempo real, esto implica que, a partir de unos datos de entrada pueda definir si se produce un evento complejo o no e inmediatamente notificarlo. Todo este proceso ha de realizarse lo más rápido posible, práticamente en tiempo real.

- **Rendimiento.** El sistema debe de ser capaz de funcionar de la forma más eficiente posible cuando la cantidad de datos de entrada sea muy elevada, garantizando así la respuesta en tiempo real.

- **Disponibilidad.** El sistema debe de ser capaz de funcionar ininterrumpidamente, esto es, funcionar los siete días de la semana las 24 horas del día.

- **Escalabilidad.** El sistema debe de ser capaz de soportar futuras ampliaciones tanto de volumen de procesamiento como de consulta.

- **Seguridad.** El sistema debe de ser capaz de mantener la integridad y seguridad de los datos que recibe y contiene.

- **Mantenibilidad.** El sistema debe de ser fácilmente mantenible, precisando el mínimo número de modificaciones o asistencia técnica con el objetivo de que su funcionamiento sea correcto e ininterrumpido.

5.3. Análisis del sistema

En los siguientes apartados mostraremos los casos de uso de las diferentes partes del sistema así como los diagramas de secuencia que los modelan de una forma más cercana a la implementación.

5.3.1. Casos de uso

Los casos de usos describen los pasos, actividades o conjunto de acciones que hemos de realizar para llevar a cabo algún proceso. Los elementos que toman parte en un caso de uso se denominan actores. Estos actores puede ser personas, entidades abstractas o sistemas externos que participan de forma activa en el caso de uso. Este proyecto se divide en dos grandes partes, por un lado la detección de eventos complejos y su notificación y por otra parte la interacción y uso de la aplicación Android por parte del usuario. Dividiremos pues los diferentes casos de uso en función de la parte del sistema con la que mejor que se identifique.

5.3.1.1. Casos de uso FIWARE

5.3.1.1.1. Caso de uso recepción datos en el sistema

- **Descripción:** La plataforma FIWARE, de forma más específica el Broker del componente Cepheus, recibe los datos de las medidas de los sensores de calidad del aire y se encarga de enviarlos al gestor de contextos Orion y al motor CEP para su análisis.
- **Actores:** Cepheus Broker (principal), Sensor (secundario), Motor CEP de Cepheus (secundario) y Orion Context Broker (secundario).
- **Precondiciones:** Todo el sistema debe estar en funcionamiento y el componente Cepheus debe conocer la estructura de los datos que va a recibir por parte de los sensores.
- **Postcondiciones:** La información relativa a los sensores, eventos simples, son almacenados en el gestor de contextos y enviados al motor CEP para su análisis.
- **Escenario principal:**
  - El sensor envía la información al sistema (recibida por el Cepheus Broker).
  - El Cepheus Broker envía estos eventos simples al motor CEP (Cepheus CEP) y al gestor de contextos Orion.
  - El motor CEP recibe los datos y comienza a analizarlos.
  - El gestor de contextos recibe y almacena el evento simple en forma de entidad.
- **Escenario alternativo:**
  - El sensor envía la información al sistema (recibida por el Cepheus Broker).

- Los datos, eventos simples, no se corresponden con ningún tipo de datos almacenado y configurado en el sistema, luego Cepheus Broker muestra un aviso por pantalla de que el tipo de dato no es reconocible y continua su funcionamiento.

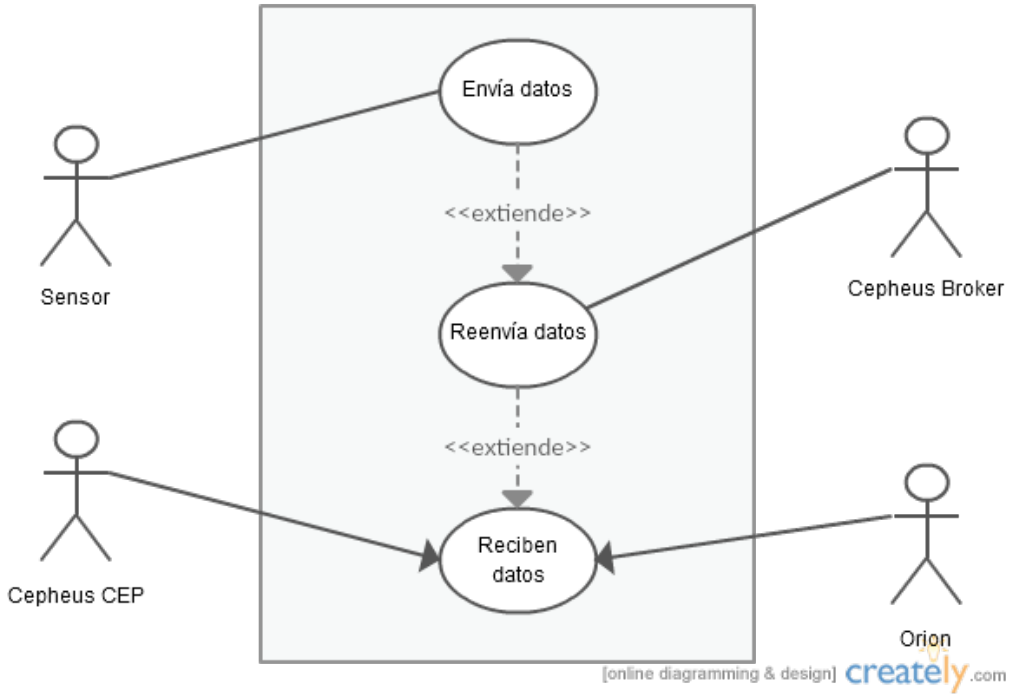

Figura 5.1.: Caso de uso recepción datos en el sistema

5.3.1.1.2. Caso de uso detección evento complejo

- **Descripción:** El motor CEP del componente Cepheus detecta un evento complejo a partir de la información que ha recibido previamente y notifica este evento complejo al Cepheus Broker que, a su vez, notificará al Orion Context Broker.
- **Actores:** Cepheus CEP (principal), Cepheus Broker (secundario) y Orion Context Broker (secundario).
- **Precondiciones:** El sistema debe de estar en funcionamiento y, al menos, debe de haber suficiente información para que detectemos un evento complejo y el patrón que dé lugar a dicho evento debe estar definido en el sistema.

- **Postcondiciones:** La información del evento complejo detectado por el motor CEP la enviamos al Broker que a su vez la envía a Orion.
- **Escenario principal:**
  - El Cepheus Broker envía los eventos simples que recibe al motor CEP (Cepheus CEP).
  - El motor CEP detecta un evento complejo a partir de una serie de patrones que hemos definido previamente.
  - El motor CEP notifica al Broker que ha detectado un evento complejo y le envía la información correspondiente al mismo.
  - El Broker hace lo propio enviando la información del evento complejo a Orion Context Broker que se encarga de registrarlo en forma de entidad.

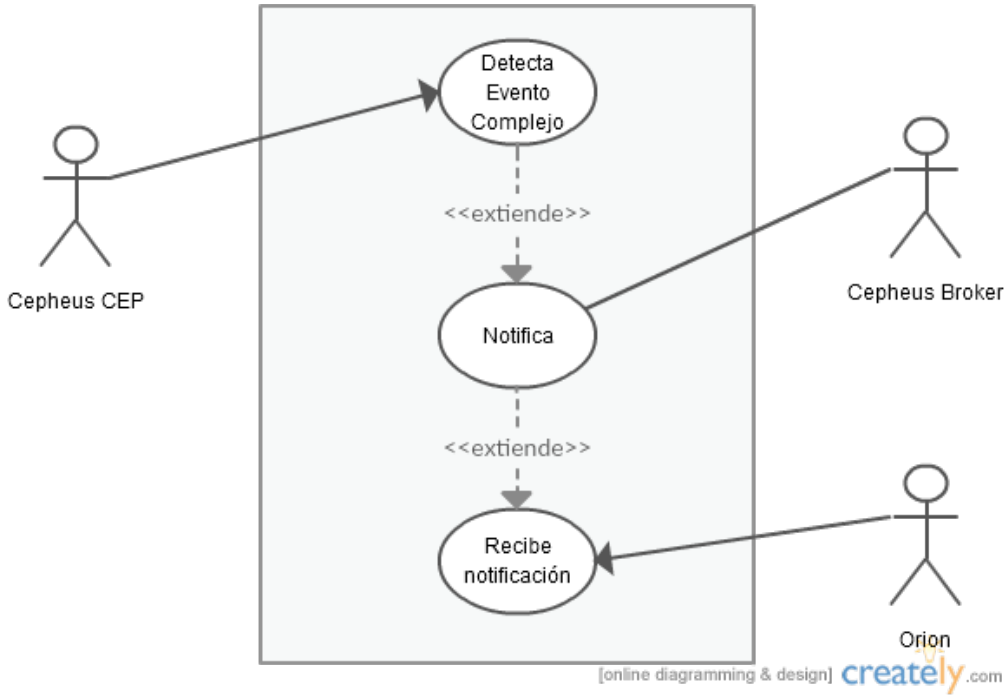

Figura 5.2.: Caso de uso detección evento complejo

5.3.1.1.3. Caso de uso suscripción a eventos complejos

- **Descripción:** Un módulo escrito en Python, a partir de ahora consumidor, se suscribe con una serie de condiciones (cambio de valor de atributo) a un tipo

de entidad concreta en Orion Context Broker (evento complejo almacenado). Orion confirma la suscripción y le indica su ID.

- **Actores:** Consumidor (principal) y Orion Context Broker (secundario).
- **Precondiciones:** El sistema debe de estar en funcionamiento.
- **Postcondiciones:** Orion Context Broker facilitará un ID de suscripción al consumidor, dicho ID identificará los valores de dicha suscripción (condiciones, periodo de validez, etc.).
- **Escenario principal:**
  - Ejecutamos el consumidor en una terminal.
  - Enviamos la petición de suscripción (comando cURL con un *payload* [49]).
  - Orion responde con el ID de suscripción.
- **Escenario alternativo (Error):**
  - En el paso 2, indicamos un *payload* incorrecto.
  - Orion notifica el error y volveríamos al primer paso.

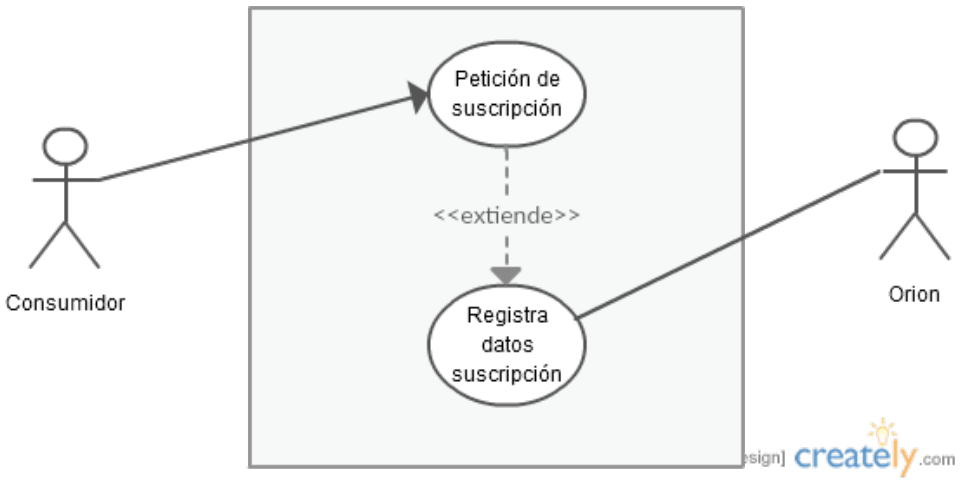

Figura 5.3.: Caso de uso suscripción a eventos complejos

5.3.1.1.4. Caso de uso notificación evento complejo a consumidor

- **Descripción:** Un módulo escrito en Python, a partir de ahora consumidor, se encuentra suscrito bajo unas determinadas condiciones a los eventos complejos de un tipo determinado. Cuando se producen dichas condiciones, Orion Context Broker notifica al consumidor de que se ha producido el evento complejo con las condiciones que el consumidor ha preestablecido.
- **Actores:** Consumidor (principal) y Orion Context Broker (secundario).
- **Precondiciones:** El sistema debe de estar en funcionamiento y el consumidor debe estar suscrito al tipo de evento complejo a Orion.
- **Postcondiciones:** Orion Context Broker notificará al consumidor de que se ha producido el evento complejo al que está suscrito y le enviará dicha información.
- **Escenario principal:**
  - Se detecta un evento complejo.
  - Orion consulta sus suscripciones activas.
  - Notifica al consumidor de que se ha producido el evento complejo al que se encuentra suscrito.
  - Las características de dicho evento complejo detectado se muestran por pantalla en la terminal del consumidor.

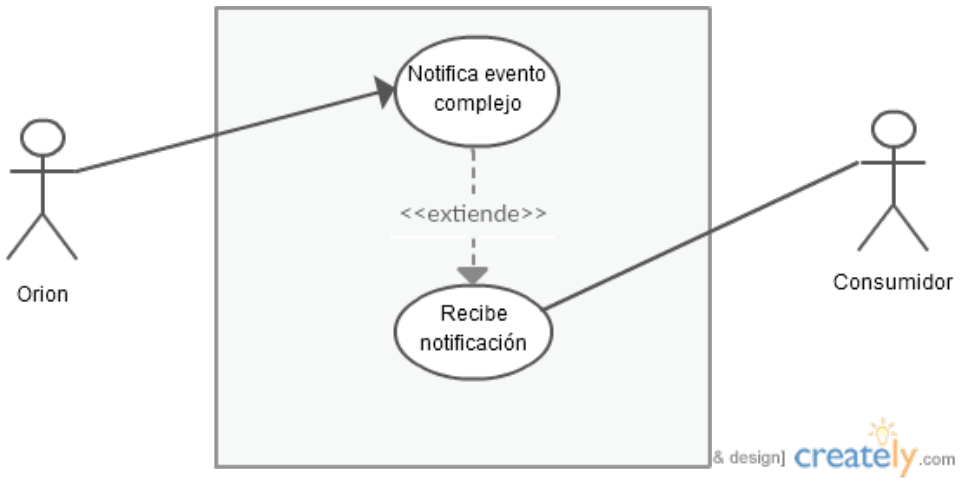

Figura 5.4.: Caso de uso notificación evento complejo a consumidor

5.3.1.1.5. Caso de uso notificación de alerta a usuario

- **Descripción:** El consumidor recibe la notificación por parte de Orion y posteriormente invoca al Servicio Web que se encargará de crear la alerta que le llegará al usuario final en su dispositivo con la aplicación Android instalada.
- **Actores:** Consumidor (principal), Servicio Web (secundario) y Usuario (secundario).
- **Precondiciones:** El sistema debe de estar en funcionamiento, el consumidor debe estar suscrito al tipo de evento complejo a Orion y debe existir un usuario con la aplicación instalada en su dispositivo.
- **Postcondiciones:** El usuario, al cual el evento complejo detectado le afecta, es notificado en su dispositivo.
- **Escenario principal:**
  - El consumidor recibe la notificación por parte de Orion de un evento complejo producido al cual se encuentra suscrito.
  - El consumidor procesa la información del evento complejo e invoca al Servicio Web que corresponda para crear la notificación.
  - El Servicio Web recibe una serie de parámetros, que es la información del evento complejo detectado, lo registra en la base de datos y crea la notificación que reciben los usuarios afectados por dicho evento complejo detectado.
  - El usuario recibe la alerta en su dispositivo.
- **Escenario alternativo (Error):** se produce alguna inconsistencia o fallo en el paso de mensajes. En tal caso el sistema seguirá su funcionamiento y este proceso será abortado en el punto que se produzca dicho error.

5.3.1.2. Casos de uso Android

5.3.1.2.1. Caso de uso registrar usuario en PushBots

- **Descripción:** El usuario, la primera vez que ejecuta la aplicación, queda registrado en el sistema de notificaciones PushBots.
- **Actores:** Usuario (principal), PushBots (secundario) y Aplicación (secundario).

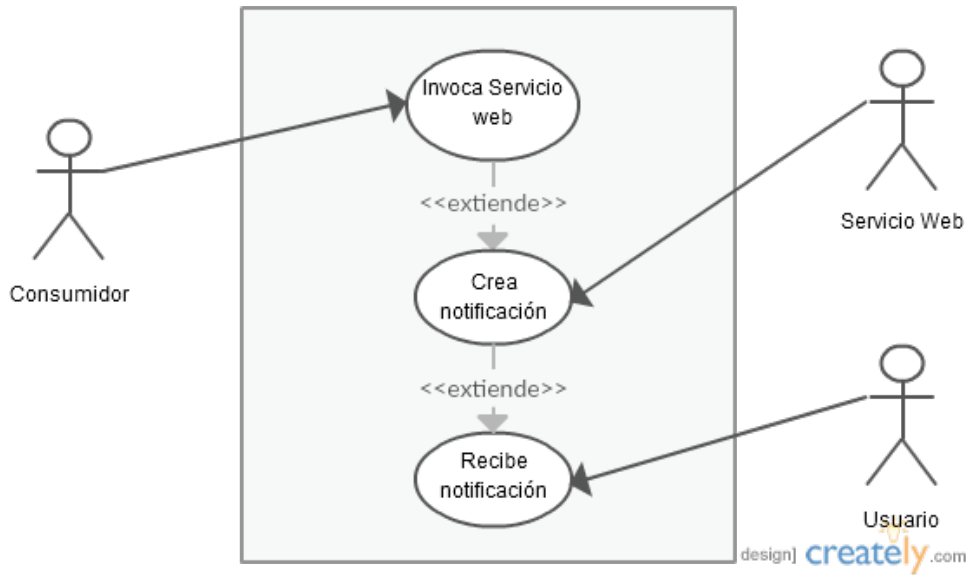

Figura 5.5.: Caso de uso notificación de alerta a usuario

- **Precondiciones:** El usuario debe disponer de la aplicación instalada en su dispositivo Android.
- **Postcondiciones:** El usuario quedará registrado en el sistema de notificaciones PushBots.
- **Escenario principal:**
  - El usuario inicia la aplicación por primera vez.
  - Se invoca la operación que registra el dispositivo de forma única en el sistema de notificaciones PushBots.
- **Escenario alernativo (Error):** se produce un error a la hora de invocar a la operación que registra al dispositivo (fallo en la conexión al servidor, sin conexión a internet, etc.). El sistema no realiza la operación de registro y, la próxima vez que el usuario ejecute la aplicación, la aplicación intentará de nuevo registrar el dispositivo en PushBots.

5.3.1.2.2. Caso de uso almacenar datos y preferencias del usuario

- **Descripción:** El usuario, la primera vez que ejecuta la aplicación, deberá rellenar un pequeño formulario en el que indicará su rango de edad, el tipo

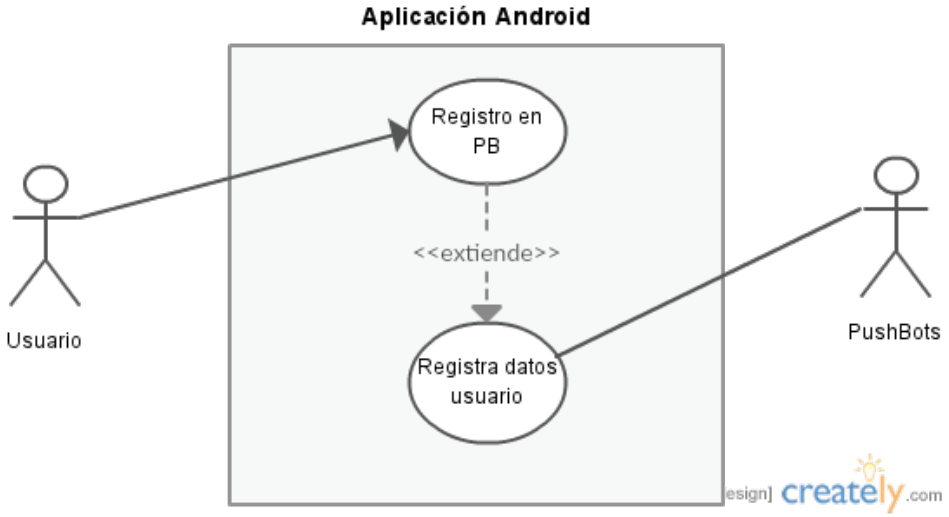

Figura 5.6.: Caso de uso registrar usuario en PushBots

de alertas del que desea ser notificado y si padece algún tipo de enfermedad respiratoria.

- **Actores:** Usuario (principal) y Aplicación (secundario).
- **Precondiciones:** El usuario debe disponer de la aplicación instalada en su dispositivo Android.
- **Postcondiciones:** Los datos y las preferencias del usuario quedarán almacenadas en su dispositivo Android.
- **Escenario principal:**
  - El usuario inicia la aplicación por primera vez.
  - Se muestra un pequeño formulario en el que el usuario deberá introducir sus datos y preferencias previamente indicados.
- **Escenario alernativo (Error):** el usuario cancela la introducción de datos y cierra la aplicación. Cuando vuelva a entrar deberá introducir los datos de nuevo.

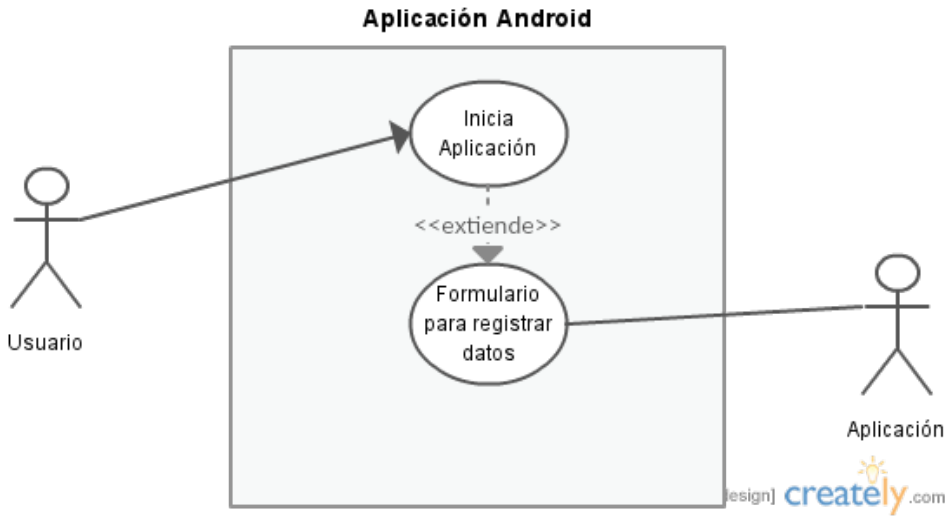

Figura 5.7.: Caso de uso almacenar datos y preferencias del usuario

5.3.1.2.3. Caso de uso consultar alertas

- **Descripción:** El usuario accede al menú y pincha sobre la opción "*Alerts detected*" que le mostrarán las últimas alertas detectadas en su ubicación.
- **Actores:** Usuario (principal) y Servicio Web de Alertas (secundario).
- **Precondiciones:** El usuario debe disponer de la aplicación instalada en su dispositivo Android.
- **Postcondiciones:** Se mostrarán las últimas alertas de calidad y, por lo tanto, la calidad del aire actual, en la ubicación del usuario.
- **Escenario principal:**
  - El usuario accede a la opción "*Alerts detected*".
  - Mediante un proceso de segundo plano, se lee su ubicación y se invoca al Servicio Web que devuelve las últimas alertas de calidad del aire para la ubicación que recibe como parámetro.
  - La información devuelta por el Servicio Web es procesada en la aplicación y se muestra al usuario en un *ListView [14].*

- **Escenarios alernativos:**
  - No hay alertas para dicha ubicación
    - El Servicio Web no devuelve información y en la aplicación no se muestra ninguna alerta, se muestra un mensaje indicando al usuario que no hay alertas registradas para su ubicación.
  - Error en algún punto del proceso
    - Se produce algún error en algún momento de este proceso (error en la base de datos, fallo de conexión).
    - No se mostrarán alertas en la aplicación y continuará su funcionamiento sin problemas.

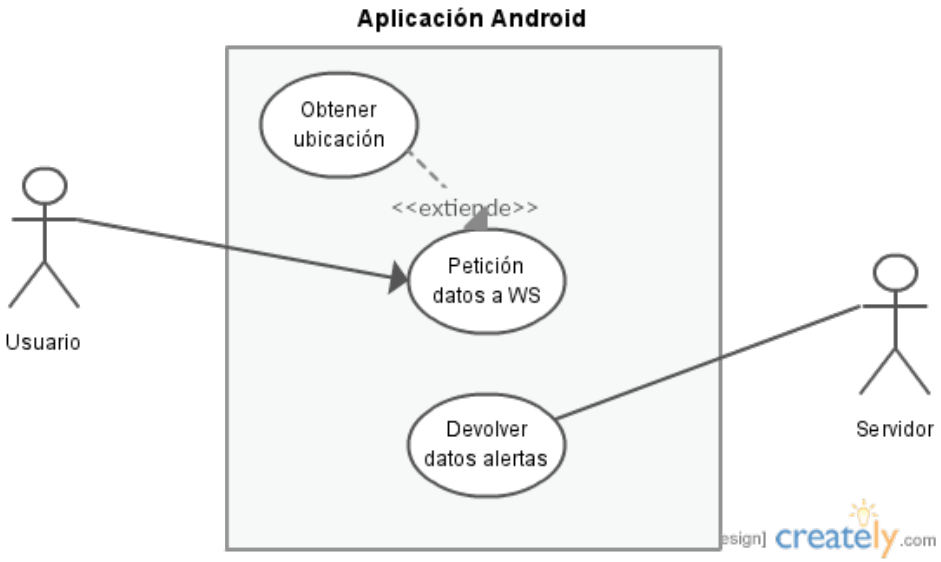

Figura 5.8.: Caso de uso consultar alertas

5.3.1.2.4. Caso de uso monitorización del usuario mediante GPS

- **Descripción:** El usuario inicia la monitorización de su contexto mediante la obtención de su localización y velocidad de movimiento gracias al sensor GPS del dispositivo. En este proceso de monitorización se le asigna un TAG al

usuario en el sistema de notificación acorde a su contexto y localización con el objetivo de ser notificado en función de su contexto personalizado.

- **Actores:** Usuario (principal), Aplicacion (secundario) y PushBots (secundario).
- **Precondiciones:** El usuario debe disponer de la aplicación instalada en su dispositivo Android y encontrarse registrado en el sistema de notificaciones de PushBots. Debe tener el sensor de ubicación (GPS) activo y conexión a internet. Para actualizar la ubicación del usuario, éste debe desplazarse 100 metros y deben haber pasado 5 minutos con respecto a su última actualización de ubicación.
- **Postcondiciones:** Se realizará la monitorización del usuario actualizando su TAG en el sistema de notificaciones correspondiente al contexto que tiene en cada instante.
- **Escenario principal:**
  - El usuario accede a la opción "*GPS Tracking*" .
  - Comprobamos que esté el GPS activo.
  - Verificamos que el usuario esté registrado en el sistema de notificaciones.
  - Obtenemos la ubicación y velocidad del usuario y, en función de estos datos, creamos un TAG que le identifique. Por ejemplo, si el usuario se encuentra en Cádiz con una velocidad superior a 20km/h, se le asigna el TAG *CadizTransport* en el sistema de notificaciones, previa eliminación del TAG que tiene actualmente asignado en el sistema.
- **Escenarios alernativos:**
  - El usuario no tiene conexión a internet
    - Mostramos un mensaje en pantalla pidiéndole al usuario que solucione sus problemas de conexión y la monitorización es abortada.
  - El usuario no tiene el GPS activo
    - Mostramos un mensaje en pantalla pidiéndole al usuario que active el GPS para iniciar la monitorización. La aplicación queda a la espera de que se active.
  - El usuario no está registrado en PushBots
    - Registramos al usuario en el sistema y la aplicación continua su tarea de monitorización.

Figura 5.9.: Caso de uso monitorización del usuario mediante GPS

5.3.1.2.5. Caso de uso establecer hábitos

- **Descripción:** El usuario registra, para uno o más días de la semana, la actividad que se encontrará realizando en los diferentes periodos de tiempo en los que dividimos el día, además de indicar la ubicación en la cual se encontrará.
- **Actores:** Usuario (principal) y Aplicación (secundario).
- **Precondiciones:** El usuario debe disponer de la aplicación instalada en su dispositivo Android.
- **Postcondiciones:** Quedará registrada la actividad que el usuario realizará en los diferentes periodos del día.
- **Escenario principal:**
  - El usuario accede a la opción "*User habits*" .
  - Mostramos el horario para el día de la semana en el cual nos encontremos.
  - El usuario pulsa sobre el botón *Edit* para modificar/añadir el horario de dicho día.

- Para cada fragmento horario en el cual hemos dividido el día, el usuario deberá indicar el tipo de actividad que se encontrará realizando (Estático, Caminando, Ejercicio o Medio de transporte).
- El usuario indicará la ubicación en la cual estará dicho día.
- El usuario guardará los cambios y volverá al paso 2.

▪ **Escenarios alernativos:**

- El usuario no rellena la información
  - En caso de que el usuario no indique que actividad estará realizando en algún fragmento horario del día o la ubicación, mostramos un mensaje en pantalla pidiéndole al usuario que solucione estos problemas y quedaremos a la espera de que vuelva a tratar de almacenar la información.

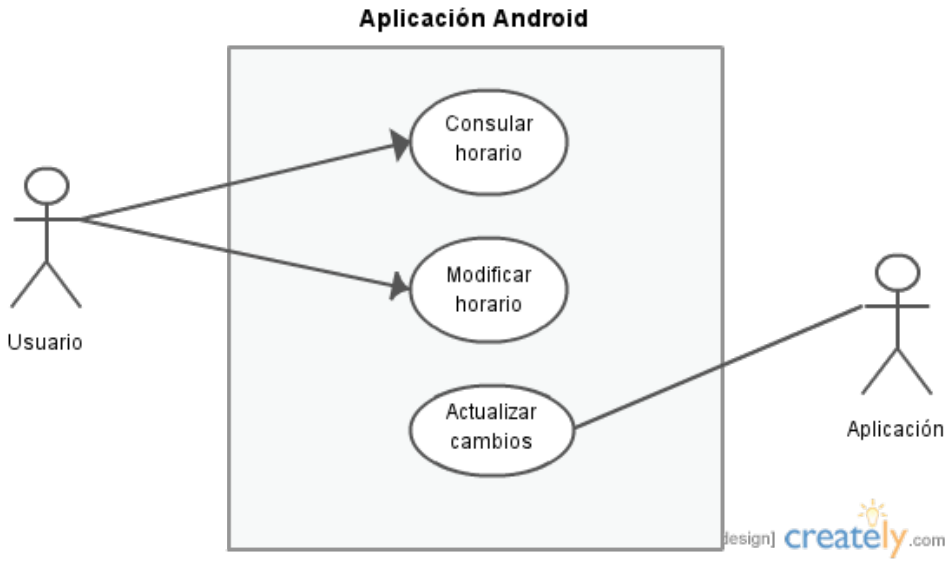

Figura 5.10.: Caso de uso establecer hábitos

5.3.1.2.6. Caso de uso monitorización del usuario mediante horario

▪ **Descripción:** El usuario inicia la monitorización de su contexto haciendo uso del horario que él previamente ha establecido en el apartado "*User habits*". En

este proceso de monitorización se le asigna un TAG al usuario en el sistema de notificación acorde a su contexto y localización con el objetivo de ser notificado en función de su contexto personalizado.

- **Actores:** Usuario (principal), Aplicacion (secundario) y PushBots (secundario).
- **Precondiciones:** El usuario debe disponer de la aplicación instalada en su dispositivo Android y encontrarse registrado en el sistema de notificaciones de PushBots. Debe tener establecido el horario para el día en el cual desea realizar la monitorización y conexión a internet. La actualización del TAG en el sistema de notificaciones se realiza una vez por cada fragmento horario en el cual hemos dividido el día.
- **Postcondiciones:** Se realizará la monitorización del usuario actualizando su TAG en el sistema de notificaciones correspondiente al contexto que tiene establecido en su horario para cada fragmento del día.
- **Escenario principal:**
  - El usuario accede a la opción "*Timetable tracking*" .
  - Comprobamos que tenga establecido el horario para el día a monitorizar.
  - Verificamos que el usuario esté registrado en el sistema de notificaciones.
  - En función de la localización y el tipo de actividad que haya establecido el usuario en su horario, creamos un TAG que le identifique. Por ejemplo, si el usuario se encuentra en Cádiz con una velocidad superior a 20km/h, se le asigna el TAG *CadizTransport* en el sistema de notificaciones, previa eliminación del TAG que tiene actualmente asignado en el sistema.
  - Este TAG permanecerá hasta el siguiente fragmento horario, en el cual actualizaremos el TAG al correspondiente con dicho fragmento horario.
- **Escenarios alernativos:**
  - El usuario no tiene el conexión a internet
    - Mostramos un mensaje en pantalla pidiéndole al usuario que solucione sus problemas de conexión y la monitorización es abortada.
  - El usuario no ha establecido un horario para ese día
    - Mostramos un mensaje en pantalla pidiéndole al usuario que debe establecer el horario para ser monitorizado de esta manera.
  - El usuario no está registrado en PushBots

- Registramos al usuario en el sistema y la aplicación continua su tare de monitorización.

- El siguiente fragmento horario no está fijado
  - Puede darse la situación que el usuario haya establecido el horario completo para el día **X** de la semana pero para el día **X+1** no lo haya definido aún. En dicho caso, cuando se de el primer fragmento horario del día **X+1** y se compruebe que no está establecida la actividad, la monitorización se cancelará automáticamente.

Figura 5.11.: Caso de uso monitorización del usuario mediante horario

5.3.1.2.7. Caso de uso recepción notificación alerta de calidad

- **Descripción:** El usuario recibe una notificación de alerta de calidad del aire creada por el Servicio Web correspondiente (invocado por el consumidor) y se muestran los datos de la alerta y un mensaje personalizado al usuario en función de su contexto.

- **Actores:** Usuario (principal) y Aplicacion (secundario).

- **Precondiciones:** El usuario debe disponer de la aplicación instalada en su dispositivo Android y encontrarse registrado en el sistema de notificaciones de PushBots y con TAG asignado correspondiente a su contexto.
- **Postcondiciones:** El usuario visualizará los datos de la alerta de calidad que hemos detectado en el sistema y le mostraremos una serie de mensajes o recomendaciones en función de su contexto.
- **Escenario principal:**
  - El usuario recibe la notificación de la alerta de calidad del aire.
  - El usuario pulsa sobre la notificación y se abre la vista de la alerta en la aplicación.
  - Desde un primer momento, la notificación recibida contendrá información relativa al contexto del usuario. Además, en función de los datos que el usuario ha establecido previamente (edad y enfermedad) se le mostrará un mensaje/recomendación acorde a su contexto y circunstancias.

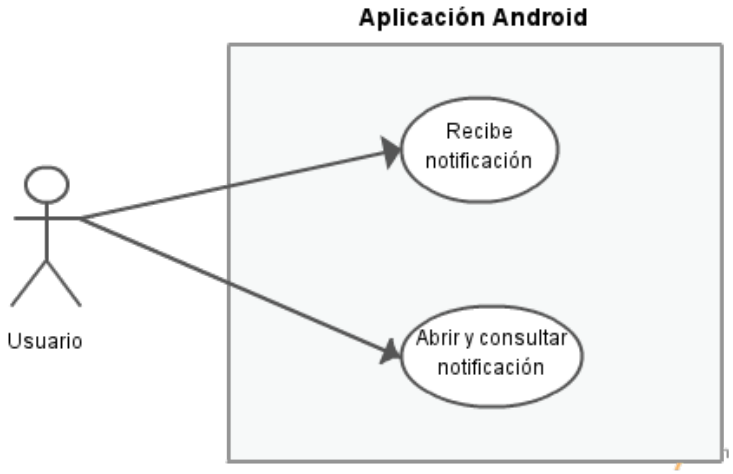

Figura 5.12.: Caso de uso recepción notificación alerta de calidad

5.3.2. Diagramas de secuencia

Al igual que hicimos con los casos de uso, diferenciaremos los diagramas de secuencia en dos partes: uno para los casos de uso que hemos definido para la plataforma

FIWARE y otro para los diferentes casos de uso que comprenden la interacción del usuario en la aplicación Android.

5.3.2.1. Diagramas de secuencia de FIWARE

A continuación, en la figura 5.13 podemos ver los diferentes casos de uso relativos a la plataforma FIWARE plasmados en un diagrama de secuencia.

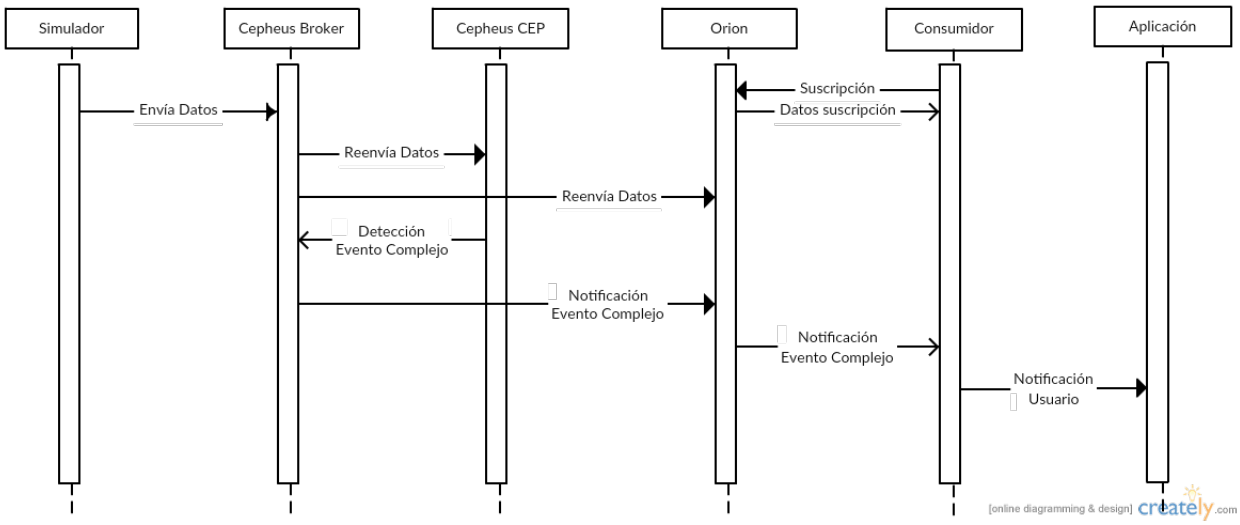

Figura 5.13.: Caso de uso recepción notificación alerta de calidad

5.3.2.2. Diagramas de secuencia de Android

A continuación, en las figuras 5.14 y 5.15 podemos ver los diferentes casos de usos relativos a la aplicación Android plasmados en diagramas de sencuencia.

5.4. Diseño del sistema

En este apartado trataremos los aspectos más específicos correspondientes al diseño del sistema, indicando las diferentes interacciones de los módulos o subsistemas en los que hemos dividido nuestro proyecto. Dividiremos este apartado en cuatro secciones; Diseño general, que comprenderá el diseño completo del sistema, comentando las diferentes partes que forman parte de él. A continuación veremos la Arquitectura hardware, que comprenderá todos los aspectos físicos que forman parte de nuestro sistema, describiendo sus características y hardware; por otro lado, Arquitectura software, en el cual comentaremos los aspectos del desarrollo del software que forma

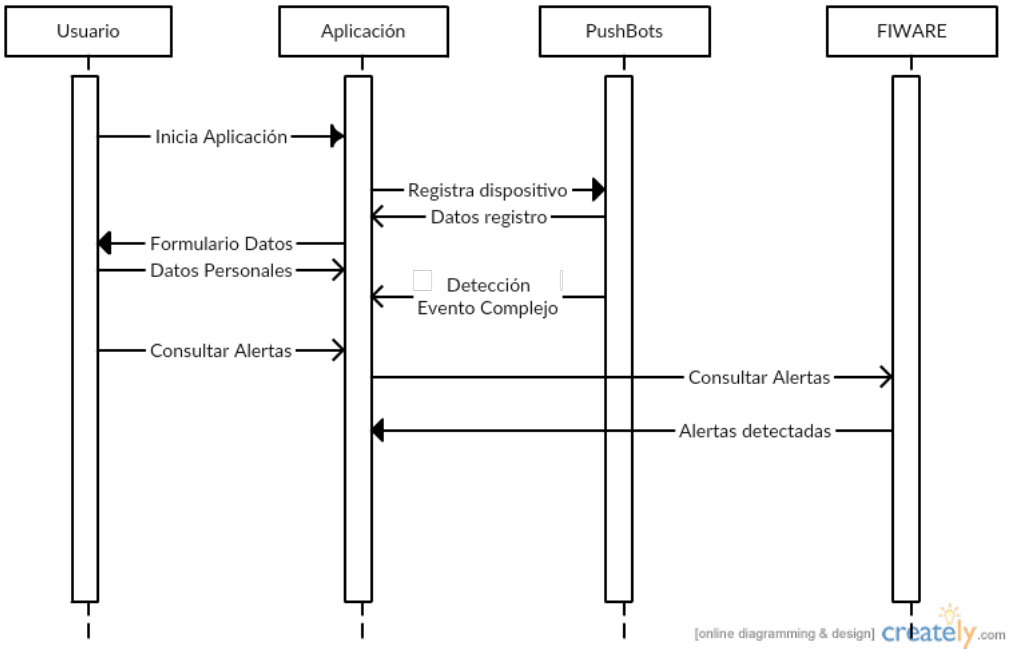

Figura 5.14.: Caso de uso recepción notificación alerta de calidad

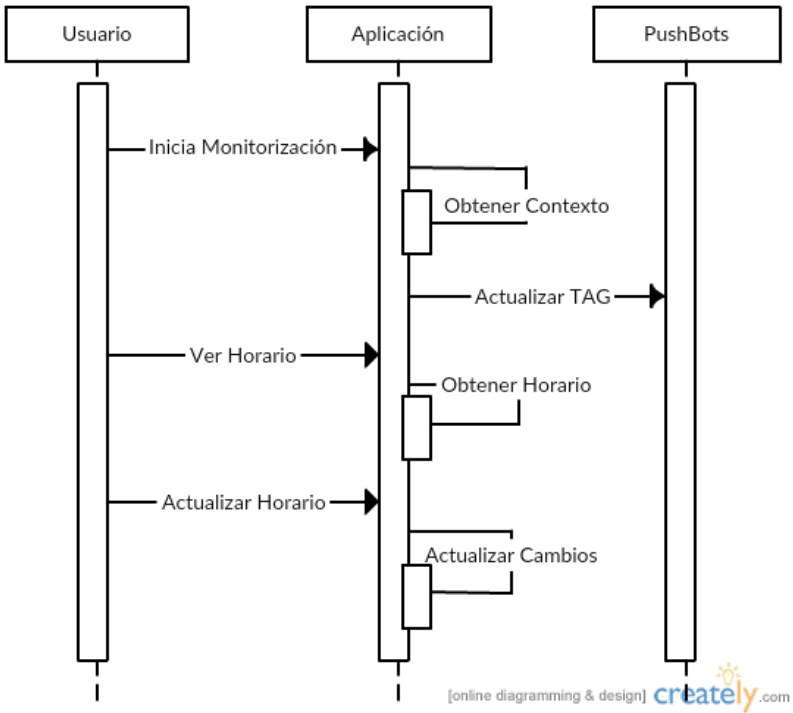

Figura 5.15.: Caso de uso recepción notificación alerta de calidad

parte del sistema y hace posible su correcto funcionamiento. Finalmente los Patrones de eventos que hemos empleado en nuestro proyecto.

5.4.1. Diseño general

En la figura 5.16 podemos apreciar el diseño del sistema de forma global. En el cual distinguimos los tres grandes elementos que forman parte del sistema. En la izquierda de la imagen podemos ver las diferentes fuentes de datos que pueden ser los sensores de calidad del aire o bien simuladores de dichos datos. En el centro de la figura podemos observar el sistema de procesamiento, compuesto por Cepheus Broker, Cepheus CEP, Orion Context Broker y el consumidor de eventos. Todos ellos forman parte y están desplegados en el servidor de FIWARE. Finalmente, podemos apreciar los diferentes dispositivos finales que almacenarán o harán uso de la información procesada, por un lado los usuarios y por otro lado la base de datos que registra las alertas.

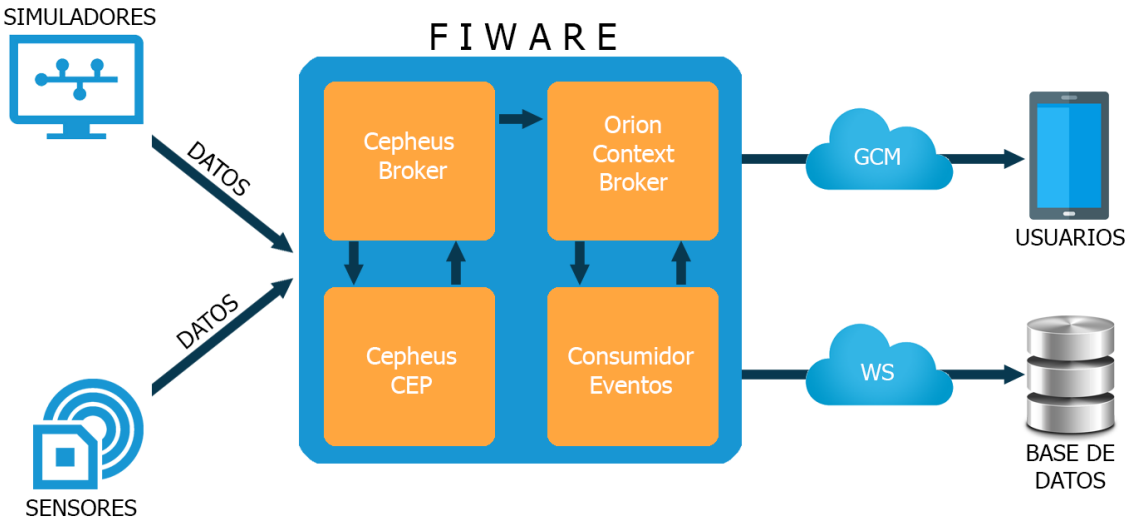

Figura 5.16.: Diseño general del proyecto

Como podemos apreciar en la figura 5.16, las fuentes de datos, en este caso los datos de los sensores o bien de los simuladores, se envían al sistema de procesamiento que se identifica con la plataforma FIWARE. Aquí se realiza el procesamiento del flujo de eventos que recibimos constatemente y, cuando se den las condiciones de detección de un patrón de eventos definido, este será notificado al consumidor de eventos que se encargará de invocar al Servicio Web correspondiente para notificar al usuario y registrarlo en una base de datos.

5.4.2. Arquitectura hardware

La arquitectura hardware del sistema se identifica con los diferentes elementos físicos que forman parte del sistema. En nuestro caso, estos elementos físicos y tangibles que forman parte en nuestro proyecto son el servidor en la nube dónde se encuentra alojado el sistema de procesamiento de información, es decir, la plataforma FIWARE y, por otro lado, el dispositivo inteligente (*smartphone, tablet, wereable...*) que tiene instalada la aplicación Android de este proyecto.

5.4.2.1. Servidor en FIWARE

En este caso, el servidor que FIWARE nos ha proporcionado durante un determinado periodo de tiempo para la realización de este proyecto cuenta con el siguiente hardware:

- 12 unidades de procesamiento virtuales (VCPUs).
- 12 GB de memoria RAM.
- 160 GB de disco duro.
- 1 IP pública.

Este hardware que nos proporciona FIWARE lo podemos dividir e instanciar en, como máximo, tres máquinas virtuales. En nuestro caso, aunque hemos hecho uso de tres máquinas virtuales con igual hardware para llevar a cabo las pruebas del funcionamiento de cada uno de los componentes empleados en diferentes equipos, solo es necesario una máquina virtual para ejecutar el sistema.

El sistema lo ejecutamos en una máquina virtual que cuenta con 2 VCPUs, 4 GB de RAM, 40 GB de disco duro y hace uso de la IP pública que nos ofrece FIWARE. En dicho sistema desplegamos los componentes necesarios para el procesamiento de eventos complejos, la gestión de contextos, módulo consumidor y finalmente el servidor apache para la base de datos y Servicios Webs.

5.4.2.2. Terminal Android

La aplicación Android, aunque se puede ejecutar en un emulador virtual sin necesidad de desplegarla en un dispositivo físico, la hemos testado y desarrollado en un terminal Android, Xperia Z1 del 2013 que cuenta con el procesador Qualcomm MSM8974 Snapdragon 800 quad-core a 2.2GHz, 2 GB de memoria RAM y 16 GB de disco duro.

Aunque el Xperia Z1 haya sido el terminal empleado para las pruebas y desarrollo de la aplicación, esta puede ser desplegada en cualquier terminal Android que cuente

con un hardware inferior debido a que la aplicación no requiere de grandes recursos ni potencia de cómputo en ningún momento, el único requisito indispensable para que la mayoría de las funcionalidades funcionen de forma óptima es que cuente con GPS y conexión a internet.

5.4.3. Arquitectura software

La arquitectura software del sistema es la más importante y tiene más peso en él, debido a que la mayor parte del trabajo de este proyecto se basa en el software. En nuestro caso, la arquitectura software del sistema la podemos dividir en dos grandes apartados; por un lado tenemos la plataforma FIWARE, en la cual realizamos la detección de eventos complejos, gestión de contextos, notificación al usuario; y por otro lado tenemos la aplicación Android, que desempeña un papel fundamental en la arquitectura del sistema ya que representa al usuario final y como éste accede a las diferentes funcionalidades desarrolladas en el proyecto.

5.4.3.1. Fuentes de datos

La obtención de los datos, o eventos simples que recibe la plataforma FIWARE, se puede realizar de dos formas:

- Sensores de calidad del aire: estos sensores pueden implementar un mecanismo que permita enviar la información vía cURL a la API REST de Cepheus Broker que se encargará de enviarlo a Orion Context Broker y Cepheus CEP, respectivamente.

- Simuladores de datos: debido a que es muy difícil que la calidad del aire varíe de forma drástica en intervalos cortos de tiempo, es necesario emplear simuladores que nos permitan generar datos en un rango determinado para que se produzcan ciertos niveles de alerta de una forma que, con datos reales, sería prácticamente imposible.

5.4.3.2. FIWARE

La plataforma FIWARE es la herramienta que hemos empleado para implementar nuestro sistema de detección de eventos complejos, gestión de contextos, suscripciones y notificaciones al usuario. Todo esto ha sido posible gracias a los diferentes componentes que ofrece FIWARE y a otros elementos que hemos instalado y desarrollado. Como pudimos observar en la figura 5.16 la estructura del sistema empleado en FIWARE está distribuida de una forma modular, interconectados entre ellos y trabajando de forma conjunta.

5.4.3.2.1. Cepheus

Es el componente encargado del procesamiento de eventos complejos. Este componente se divide a su vez en dos módulos, el Cepheus Broker y el Cepheus CEP.

- Cepheus Broker es el encargado de recibir la información en forma de eventos simples y transmitirlos al motor CEP para su procesamiento y otros componentes del sistema como Orion Context Broker. Además, realiza la comunicación del motor CEP y sus avisos, como el de un evento complejo detectado, con otros componentes. Podríamos definir al Broker como el encargado de distribuir y conducir el paso de información por el componente Cepheus.

- Cepheus CEP es el motor de procesamiento de eventos complejos del componente Cepheus. Está basado en Esper. Recibe flujos constantes de eventos por parte del Broker de Cepheus y se encarga de procesarlos y detectar eventos complejos previamente definidos en el fichero de configuración.

El fichero de configuración, generalmente config.json, es un fichero que contiene la configuración con la que el componente Cepheus CEP va a trabajar. Está definido en formato JSON [62] y compuesto por tres elementos fundamentales.

1. Estructura eventos simples. Es decir, la definición de los tipos de eventos que el motor CEP va a ser capaz de recibir y procesar.

2. Estructura eventos complejos. Es decir, el esquema o estructura que seguirán los diferentes eventos complejos detectados y generados por el motor CEP en función de los patrones.

3. Patrones de eventos complejos. Están escritos en una sintaxis muy similar a Esper, solo con algunas diferencias (como la necesidad de asignar un ID al patrón). Los patrones especifican unas condiciones que deben cumplirse para que se produzca un evento complejo.

5.4.3.2.2. Orion Context Broker

Orion Context Broker lo podríamos catalogar como el cerebro de nuestra aplicación. Es un gestor de contextos, el principal de toda la plataforma FIWARE, y complementa perfectamente al componente que hemos visto anteriormente, Cepheus.

Orion permite una funcionalidad muy importante que es la suscripción a tipos de entidades, que en nuestro caso serán eventos complejos detectados. Un consumidor se suscribe a Orion con una serie de condiciones sobre un tipo de entidad concreto. Cuando dichas condiciones se suceden (por ejemplo, el atributo $\boldsymbol{X}$ del tipo de evento complejo $\boldsymbol{Y}$ ha cambiado de valor) Orion notifica a todos los consumidores que se

encuentran suscritos, bajo esa condición, a ese tipo de evento complejo. Esto permite una reacción inmediata y asegura el poder responder a este evento en tiempo real.

Otra característica importante de Orion es su API REST que permite realizar operaciones como añadir, modificar, consultar o eliminar información almacenada fácilmente. Orion no precisa de una configuración previa como Cepheus, si no que, una vez que está en funcionamiento, está listo para recibir información, modificarla, consultarla o eliminarla, siempre que se respete su estructura y modelo de datos.

5.4.3.2.3. Consumidor de eventos

Este módulo desempeña un papel fundamental en el funcionamiento del sistema ya que es el encargado de realizar la suscripción a Orion e invocar a los diferentes Servicios Web que crean notificaciones para avisar a los usuarios. Es un código en Python, ejecutable en la terminal.

El código utilizado en este proyecto es una adaptación del código original [12] desarrollado por Fermin Galán Márquez (miembro del equipo desarrollador de Orion Context Broker), que, con una serie de modificaciones, permite obtener la información del evento complejo detectado, tratarla, e invocar al Servicio Web correspondiente con el objetivo de crear la notificación que le llegará al usuario en su dispositivo Android.

5.4.3.3. Dispositivos finales

Dentro de los diferentes dispositivos finales que recibirán los datos procesados y los diferentes eventos complejos detectados, podemos distinguir el servidor Apache que contendrá la base de datos MySQL que registrará las alertas detectadas y, por otro lado, la aplicación Android que empleará el usuario final.

5.4.3.3.1. Servidor Apache

Finalmente, otro de los apartados más importante es el servidor Apache LAMPP que ejecutamos en el sistema, que nos permite, mediante el uso de su sistema gestor de bases de datos phpMyAdmin, almacenar los eventos complejos detectados en registros de una base de datos. La base de datos que se encarga de registrar las alertas detectadas está compuesta por una tabla que presenta la estructura que podemos contemplar en la figura 5.17.

Por otro lado, gracias a que el servidor puede ejecutar scripts escritos en PHP, lo usamos para desarrollar una serie de Servicios Webs que se encargan de crear y notificar al usuario. Hemos desarrollado e implementado los siguientes Servicios Webs de forma individual:

- Alert

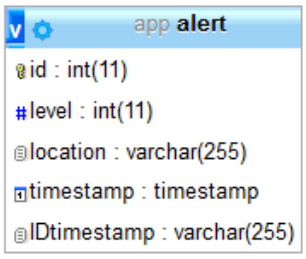

Figura 5.17.: Tabla "*alerts*"

- Este Servicio Web recibe por parámetros el nivel de alerta detectado, la localización y el instante en el que se ha detectado dicha alerta. Con estos tres parámetros en primer lugar realizaremos el registro en la base de datos MySQL del evento complejo detectado, en este caso la alerta por el nivel de calidad del aire. A continuación realizaremos la personalización de la alerta en función del nivel y añadiremos el mensaje específico que afecta a cada contexto de usuario.

- getAlerts

  - Este Servicio Web no recibe ningún parámetro. Nos devuelve, en formato JSON, el conjunto de las alertas detectadas, sin realizar ningun filtro por localización, simplemente volcamos la información contenida en la base de datos y la devolvemos.

- getAlertsByLocation

  - Este Servicio Web recibe por parámetro la ubicación del usuario o sobre la cual realizamos la consulta. Su estructura es completamente similar al del Servicio Web descrito anteriormente, solo que en este caso filtraremos los registros de la base de datos, obteniendo las alertas del nivel de calidad del aire que afecten a la localización sobre la cual realizamos la consulta.

5.4.3.4. Android

Dentro de Android podemos distinguir muchas clases y funcionalidades. A continuación describiremos las funcionalidades más representativas y, en el apartado de implementación, veremos el código fuente de las más importantes de toda la aplicación Android.

5.4.3.4.1. FragmentGPS

Este fragmento es el que se encarga de la monitorización del usuario haciendo uso del GPS. Es la funcionalidad más importante de la aplicación Android ya que se encarga de determinar el contexto del usuario en función de los datos leídos del GPS y, una vez conoce cual es el contexto, actualiza el TAG del usuario en el sistema de notificaciones, si procede. En todo momento se hacen una serie de comprobaciones como asegurarnos de que el GPS está activado o que el usuario tiene activada la opción de recibir alertas de cierto tipo, para así evitar crear un TAG que le haga ser notificado de ciertos eventos complejos de los cuales no desea ser notificado o no corresponde serlo.

5.4.3.4.2. FragmentTimeTable

Este fragmento es muy similar al anterior y realiza la misma funcionalidad, solo que en este caso la monitorización se hace mediante el uso de un horario que el usuario previamente ha establecido. Este horario divide en día en varios periodos que se identifican con los que, para la mayoría de la población, reflejan los intervalos de actividad en un día (de 00:00 a 08:00; de 08:00 a 14:00; de 14:00 a 16:00; de 16:00 a 20;00; de 20:00 a 00:00) junto a la ubicación donde se encontrará dicho día. Como hemos comentado, el objetivo es el mismo, ir estableciendo el TAG correspondiente al contexto del usuario en cada periodo de tiempo correspondiente. Para pasar actualizar el TAG existente con el correspondiente al momento en el que nos encontremos, hacemos uso de la clase AlarmManager que hace posible establecer disparadores a ciertas horas y bajo ciertas condiciones. En nuestro caso, cada vez que comienza un nuevo fragmento horario, el TAG se actualiza en el sistema de notificaciones con el TAG que corresponde a ese fragmento horario.

5.4.3.4.3. FragmentSettings

Este fragmento es el que permite al usuario establecer sus datos y preferencias. En él, el usuario rellenará un pequeño formulario en el que establecerá:

- Si padece o no alguna enfermedad respiratoria.
- El rango de edad al que pertenece (joven, adulto o anciano).
- Las notificaciones que desea recibir, es decir, los eventos complejos que detectamos en la plataforma FIWARE y que notificamos posteriormente.

En función de los parámetros establecidos en este fragmento, las notificaciones de eventos complejos mostrarán información acorde a las circusntancias del usuario, es decir, un usuario con problemas respiratorios tendrá una recomendación para un

nivel de alerta 5 de calidad del aire que un usario sin problemas respiratorios no tendría. La información será modificable en cualquier momento.

5.4.3.4.4. AlarmReceiver

Esta clase complementa al fragmento FragmentTimeTable que hemos visto anteriormente. Esta clase es la que se ejecuta cada vez que el disparador que definimos con la clase AlarmManager es detectado. La funcionalidad de este fragmento es actualizar el TAG en el sistema de notificaciones del usuario que está haciendo uso de la monitorización por horario. Para ello, consulta el horario y establece el TAG que corresponde al usuario en dicho instante. Hay que notar que esta clase solo se ejecutará una vez por cada fragmento horario. Una vez establecido el TAG, la misma clase se encarga de configurar el disparador para que se ejecute en el siguiente intervalo de tiempo correspondiente.

5.4.4. Patrones de eventos

Finalmente, otros de los apartados a notar en el aspecto de diseño son los diferentes patrones de eventos empleados para la detección de eventos complejos. Aunque en FIWARE haya pequeñas modificaciones para la implementación de los patrones, en este apartado los veremos en el lenguaje EPL.

5.4.4.1. Nivel de calidad del aire

Para implementar el evento complejo que se encarga de detectar la calidad del aire por localización es necesario definir una serie de patrones. En primer lugar distinguimos la estructura de la información que recibiremos de los sensores:

```
AirMeasurement
(sensorId, location, timestamp, pm2_5,
 pm10, o3, no2, so2, co, temp, humidity);
```

Debido a que tenemos diferentes contaminantes y, cada uno de ellos, se debe de agrupar en ventanas temporales específicas, podemos distinguir tres agrupaciones horarias:

```
NO2 y SO2 --> Periodos de 1 hora.
O3 y CO --> Periodos de 8 horas.
PM10 y PM2,5 --> Periodos de 24 horas.
```

Por lo tanto, para cada sensor existente llevaremos agrupaciones en pares de contaminantes en función de sus ventanas temporales dónde calcularemos su media. Una

vez conocemos el valor medio de cada contaminante en cada sensor, es necesario saber el valor medio de cada uno de los contaminantes para una localización dada, luego el siguiente paso es agrupar el valor de cada contaminante por localización, obteniendo así la media del contaminante en cada localización.

Llegados a este punto es necesario evaluar el valor de los contaminantes para inferir los niveles AQI detectados. Por lo tanto, definimos seis patrones de eventos AQI, uno por cada nivel AQI existente, con los valores correspondientes al valor del contaminante en cada rango AQI. De esta forma, cuando calculemos el valor medio de los contaminantes en una localización, detectaremos al instante los diferentes niveles AQI que se producirán debido a esos valores de contaminantes.

El último paso es agrupar los valores AQI producidos. Puede ser que para un mismo conjunto de datos, se produzcan un nivel 1 y un nivel 5 de AQI. En este caso, el valor AQI real será el mayor de los valores AQI en dicho instante, siendo por lo tanto AQI nivel 5. Para determinar esto, es necesario agrupar los diferentes eventos AQI que hayamos detectado por localización y notificar el de mayor valor, siendo este el verdadero valor AQI.

5.5. Implementación

Tras ver el diseño de las diferentes partes que componen nuestro proyecto, vamos a profundizar en los aspectos más relevantes, a nivel de código e implementación, de aquellas partes del proyecto que suponen una especial mención debido a su funcionalidad o complejidad.

5.5.1. Cepheus

Como hemos comentado anteriormente, para el correcto funcionamiento de Cepheus es necesario definir previamente tanto los eventos simples que recibiremos, como los eventos complejos que serán detectados por los patrones de eventos. A continuación mostraremos, por partes, el contenido del fichero de configuración que contiene esta información que acabamos de mencionar.

En primer lugar tenemos la estructura de los eventos simples de entrada, es decir, la estructura de la información que recibiremos de los sensores o fuentes de información. En este fragmento de código observamos el campo *in*, el cual es un array de objetos JSON, en el cual cada objeto representa un tipo de evento simple que podemos recibir. La estructura de cada evento está formada por el *id*, el *type* (el tipo), el campo *isPattern* indica si el ID sigue algún tipo de patrón (en este caso es verdadero ya que los eventos de entrada tendrán diferentes fuentes de origen: *AirMeasurementCadiz, AirMeasurementSevilla, ...*), el campo *providers* indica la fuente

de origen que proporciona los datos (se puede suprimir) y finalmente *attributes* que es un array que lista los diferentes atributos y su tipo.

```
{
        "host":"http://localhost:8080",
        "in":[
                {
                  "id":"AirMeasurement.*",
                  "type":"AirMeasurement",
                  "isPattern":true,
                  "providers":[
                        "http://localhost:8081"
                  ],
                  "attributes":[
                    {"name":"sensorId", "type":"string"},
                    {"name":"location", "type":"string"},
                    {"name":"pm2_5", "type": "double"},
                    {"name":"pm10", "type": "double"},
                    {"name":"o3", "type": "double"},
                    {"name":"no2", "type": "double"},
                    {"name":"so2", "type": "double"},
                    {"name":"co", "type": "double"},
                    {"name":"temperature", "type":"double"},
                    {"name":"humidity", "type": "double"}
                  ]
            }
        ],
```

A continuación tenemos la definición de los diferentes eventos complejos. La estructura es prácticamente la misma que la de los eventos simples, solo que en este caso el campo *providers* lo sustituimos por *brokers*, que indica los brokers que van a ser notificados del evento detectado. En nuestro caso especificamos el puerto de escucha de Orion Context Broker para notificarlo.

```
"out":[
    {
      "id":"O3AndCOAvg1",
      "type":"O3AndCOAvg",
      "brokers":[
        {
          "url":"http://localhost:1026"
```

```
      }
    ],
    "attributes":[
      { "name": "location", "type": "string" },
      { "name": "o3_avg", "type": "double" },
      { "name": "co_avg", "type": "double" },
      { "name": "sensorId", "type": "string" },
      { "name": "timestamp", "type": "string" }
    ]
  },
  {
    "id":"PM10AndPM25Avg1",
    "type":"PM10AndPM25Avg",
    "brokers":[
      {
        "url":"http://localhost:1026"
      }
    ],
    "attributes":[
      { "name": "location", "type": "string" },
      { "name": "pm10_avg", "type": "double" },
      { "name": "pm2_5_avg", "type": "double" },
      { "name": "sensorId", "type": "string" },
      { "name": "timestamp", "type": "string" }
    ]
  },
  {
    "id":"SO2AndNO2Avg1",
    "type":"SO2AndNO2Avg",
    "brokers":[
      {
        "url":"http://localhost:1026"
      }
    ],
    "attributes":[
      { "name": "location", "type": "string" },
      { "name": "so2_avg", "type": "double" },
      { "name": "no2_avg", "type": "double" },
      { "name": "sensorId", "type": "string" },
      { "name": "timestamp", "type": "string" }
```

```
  ]
},
{
  "id":"PollutantAvgByLocation1",
  "type":"PollutantAvgByLocation",
  "brokers":[
    {
      "url":"http://localhost:1026"
    }
  ],
  "attributes":[
    { "name":"location", "type": "string"},
    { "name":"kindPollutant", "type": "integer" },
    { "name":"val", "type": "double"},
    { "name":"timestamp", "type": "string" }
  ]
},
{
  "id":"AQILevels1",
  "type":"AQILevels",
  "brokers":[
    {
      "url":"http://localhost:1026"
    }
  ],
  "attributes":[
    { "name":"alertLevel", "type":"integer" },
    { "name":"kindPollutant", "type": "integer" },
    { "name":"val", "type": "double" },
    { "name":"location", "type": "string" },
    { "name":"timestamp", "type": "string" }
  ]
},
{
  "id":"AQIByLocation1",
  "type":"AQIByLocation",
  "brokers":[
    {
      "url":"http://localhost:1026"
    }
```

```
      ],
      "attributes":[
        { "name":"location", "type": "string" },
        { "name":"alertLevel", "type": "integer" },
        { "name":"timestamp", "type": "string" }
      ]
    }
  ],
```

Finalmente, tenemos el campo *statements* en el cual definimos los patrones de eventos que deseamos detectar. Como ya hemos comentado, Cepheus CEP funciona bajo Esper, luego la sintaxis para escribir los patrones es muy similar a EPL de Esper con algunas ligeras modificaciones. Cabe destacar que haciendo uso del operador de concatenación || junto al *timestamp* logramos la creación del ID único para cada evento, necesario en la definición de eventos complejos en el componente Cepheus de FIWARE.

```
"statements":[
"INSERT INTO O3AndCOAvg
SELECT avg(e.o3) as o3_avg, avg(e.co) as co_avg,
e.location as location, e.sensorId as sensorId,
current_timestamp.toString() as timestamp
FROM pattern [every e = AirMeasurement].win:time(8 hour)
GROUP BY e.sensorId",

"INSERT INTO PM10AndPM25Avg
SELECT avg(e.pm10) as pm10_avg, avg(e.pm2_5) as pm2_5_avg,
e.location as location, e.sensorId as sensorId,
current_timestamp.toString() as timestamp
FROM pattern [every e = AirMeasurement].win:time(24 hour)
GROUP BY e.sensorId",

"INSERT INTO SO2AndNO2Avg
SELECT avg(e.so2) as so2_avg, avg(e.no2) as no2_avg,
e.location as location, e.sensorId as sensorId,
current_timestamp.toString() as timestamp
FROM pattern [every e = AirMeasurement].win:time(1hour)
GROUP BY e.sensorId",

"INSERT INTO PollutantAvgByLocation
SELECT e.location || current_timestamp.toString() as id,
```

```
1 as kindPollutant, e.location as location, max(e.o3_avg) as val,
current_timestamp.toString() as timestamp
FROM pattern [every e = O3AndCOAvg].win:time_batch(5 min)
GROUP BY e.location HAVING max(e.o3_avg) is not null",

"INSERT INTO PollutantAvgByLocation
SELECT e.location || current_timestamp.toString() as id,
2 as kindPollutant, e.location as location, max(e.co_avg) as val,
current_timestamp.toString() as timestamp
FROM pattern [every e = O3AndCOAvg].win:time_batch(5 min)
GROUP BY e.location HAVING max(e.co_avg) is not null",

"INSERT INTO PollutantAvgByLocation
SELECT e.location || current_timestamp.toString() as id,
3 as kindPollutant, e.location as location, max(e.pm10_avg) as val,
current_timestamp.toString() as timestamp
FROM pattern [every e = PM10AndPM25Avg].win:time_batch(5 min)
GROUP BY e.location HAVING max(e.pm10_avg) is not null",

"INSERT INTO PollutantAvgByLocation
SELECT e.location || current_timestamp.toString() as id,
4 as kindPollutant, e.location as location, max(e.pm2_5_avg) as val,
current_timestamp.toString() as timestamp
FROM pattern [every e = PM10AndPM25Avg].win:time_batch(5 min)
GROUP BY e.location HAVING max(e.pm2_5_avg) is not null",

"INSERT INTO PollutantAvgByLocation
SELECT e.location || current_timestamp.toString() as id,
5 as kindPollutant, e.location as location, max(e.so2_avg) as val,
current_timestamp.toString() as timestamp
FROM pattern [every e = SO2AndNO2Avg].win:time_batch(5 min)
GROUP BY e.location HAVING max(e.so2_avg) is not null",

"INSERT INTO PollutantAvgByLocation
SELECT e.location || current_timestamp.toString() as id,
6 as kindPollutant, e.location as location, max(e.no2_avg) as val,
current_timestamp.toString() as timestamp
FROM pattern [every e = SO2AndNO2Avg].win:time_batch(5 min)
GROUP BY e.location HAVING max(e.no2_avg) is not null",
```

```
"INSERT INTO AQILevels
SELECT 1 as alertLevel, e.kindPollutant as kindPollutant,
e.timestamp as timestamp, e.location as location, e.val as val,
e.id as id
FROM pattern [every e = PollutantAvgByLocation(
(kindPollutant = 1 and val >= 0 and val <= 0.064)
or  (kindPollutant = 2 and val >= 0 and val <= 4.4)
or  (kindPollutant = 3 and val >= 0 and val <= 54)
or  (kindPollutant = 4 and val >= 0 and val <= 15.4)
or  (kindPollutant = 5 and val >= 0 and val <= 0.034))]",

"INSERT INTO AQILevels
SELECT 2 as alertLevel, e.kindPollutant as kindPollutant,
e.timestamp as timestamp, e.location as location, e.val as val,
e.id as id
FROM pattern [every e = PollutantAvgByLocation(
(kindPollutant = 1 and val >= 0.065 and val <= 0.084)
or  (kindPollutant = 2 and val >= 4.5 and val <= 9.4)
or  (kindPollutant = 3 and val >= 55 and val <= 154)
or  (kindPollutant = 4 and val >= 15.5 and val <= 40.4)
or  (kindPollutant = 5 and val >= 0.035 and val <= 0.144))]",

"INSERT INTO AQILevels
SELECT 3 as alertLevel, e.kindPollutant as kindPollutant,
e.timestamp as timestamp, e.location as location, e.val as val,
e.id as id
FROM pattern [every e = PollutantAvgByLocation(
(kindPollutant = 1 and val >= 0.085 and val <= 0.104)
or  (kindPollutant = 2 and val >= 9.5 and val <= 12.4)
or  (kindPollutant = 3 and val >= 155 and val <= 254)
or  (kindPollutant = 4 and val >= 40.5 and val <= 65.4)
or  (kindPollutant = 5 and val >= 0.145 and val <= 0.224))]",

"INSERT INTO AQILevels
SELECT 4 as alertLevel, e.kindPollutant as kindPollutant,
e.timestamp as timestamp, e.location as location, e.val as val,
e.id as id
FROM pattern [every e = PollutantAvgByLocation(
(kindPollutant = 1 and val >= 0.105 and val <= 0.124)
or  (kindPollutant = 2 and val >= 12.5 and val <= 15.4)
```

```
or  (kindPollutant = 3 and val >= 255 and val <= 354)
or  (kindPollutant = 4 and val >= 65.5 and val <= 150.4)
or  (kindPollutant = 5 and val >= 0.225 and val <= 0.304))]",

"INSERT INTO AQILevels
SELECT 5 as alertLevel, e.kindPollutant as kindPollutant,
e.timestamp as timestamp, e.location as location, e.val as val,
e.id as id
FROM pattern [every e = PollutantAvgByLocation(
(kindPollutant = 1 and val >= 0.125 and val <= 0.374)
or  (kindPollutant = 2 and val >= 15.4 and val <= 30.4)
or  (kindPollutant = 3 and val >= 355 and val <= 424)
or  (kindPollutant = 4 and val >= 150.5 and val <= 205.4)
or  (kindPollutant = 5 and val >= 0.305 and val <= 0.604)
or  (kindPollutant = 6 and val >= 0.65 and val <= 1.24))]",

"INSERT INTO AQILevels
SELECT 6 as alertLevel, e.kindPollutant as kindPollutant,
e.timestamp as timestamp, e.location as location, e.val as val,
e.id as id
FROM pattern [every e = PollutantAvgByLocation(
(kindPollutant = 1 and val >= 0.375)
or  (kindPollutant = 2 and val >= 30.5)
or  (kindPollutant = 3 and val >= 425)
or  (kindPollutant = 4 and val >= 250.5)
or  (kindPollutant = 5 and val >= 0.605)
or  (kindPollutant = 6 and val >= 1.25))]",

"INSERT INTO AQIByLocation
SELECT 'AQIByLocation' || e.location as id, e.location as location,
max(e.alertLevel) as alertLevel, current_timestamp.toString() as timestamp
FROM pattern [every e = AQILevels].win:time_batch(5 sec)
GROUP BY e.location HAVING max(e.alertLevel) is not null"

]
}
```

5.5.2. Consumidor de eventos

El consumidor, como ya hemos comentado en apartados anteriores, se encarga de suscribirse en Orion Context Broker a los eventos complejos de un tipo determinado.

Cuando se detecta el evento complejo, el consumidor será notificado por Orion y tendrá que realizar una serie de acciones. En nuestro caso, tenemos varios consumidores, uno por cada tipo de evento complejo detectado. La estructura de los consumidores es muy similar, solo el tratamiento de los atributos del evento complejo a detectar es lo que cambia en el código. El código de nuestro consumidor Python está basado en el original desarrollado por Fermin Galán Márquez para Orion Context Broker.

A continuación mostramos el fragmento de código que se encarga de obtener los datos de la alerta de calidad del aire detectada e invocar al Servicio Web. En primer lugar mostramos el JSON por pantalla y le realizamos una serie de modificaciones para poder tratarlo. A continuación obtenemos los valores de los atributos que buscamos (calidad del aire, instante y ubicación), con estos valores, realizamos la llamada al Servicio Web que se encarga de crear la notificación que el usuario final recibirá en su dispositivo.

```
if verbose:
        print s
        valueJSON = valueJSON.replace('\n','')
        valueJSON = valueJSON.replace(' ','')
        import json
        j = json.loads(valueJSON)
        print "Nivel de alarma %s" %
            j['contextResponses'][0]['contextElement']['attributes'][0]['value']
        print "Localidad %s" %
            j['contextResponses'][0]['contextElement']['attributes'][1]['value']
        print "Timestamp %s" %
            j['contextResponses'][0]['contextElement']['attributes'][2]['value']
        nivel =
            j['contextResponses'][0]['contextElement']['attributes'][0]['value']
        localidad =
            j['contextResponses'][0]['contextElement']['attributes'][1]['value']
        idtime =
            j['contextResponses'][0]['contextElement']['attributes'][2]['value']
        url =
            "http://130.206.117.253/SlimWS/app/alert.php/addAlert/%i/%s/%s"
            % (int(nivel),localidad,idtime)
        print "Realizando peticion a %s" % url
        import urllib2
        urllib2.urlopen(url).read()
        print "Notificacion creada"
```

5.5.3. Servicios Web

Como hemos visto anteriormente, tenemos diferentes Servicios Web que realizan diferentes funcionalidades. A continuación mostraremos el código fuente de aquellos que realizan funciones más notables en nuestro sistema.

5.5.3.1. Alert

Este Servicio Web registra una alerta detectada, recibiendo el nivel de alerta, la localización y el instante por parámetros, y crea la notificación, realizando la personalización del contexto si el nivel de alerta lo requiere, que afectará a los usuarios de dicha localización. La implementación de este servicio se encuentra disponible en el Anexo D.1.1.

5.5.3.2. GetAlertsByLocation

Este Servicio Web devuelve las últimas cinco alertas detectadas para una ubicación que recibimos por parámetro. Obtiene la información de la base de datos, la procesa, la codifica en JSON y la devolvemos. La implementación de este servicio se encuentra disponible en el Anexo D.1.2.

5.5.4. Android

Debido a que el código fuente de la aplicación Android es bastante extenso, mostraremos los códigos fuentes de las funcionalidades más notables en nuestra aplicación Android.

5.5.4.1. FragmentGPS

Como ya vimos en el apartado de diseño esta clase de nuestra aplicación se encarga de la monitorización del usuario haciendo uso del GPS. En el código incluimos comentarios explicativos para su compresión. La implementación de este servicio se encuentra disponible en el Anexo D.2.1.

5.5.4.2. AlarmReceiver

Esta clase es la encargada de ejecutarse cuando el disparador configurado con AlarmManager se detecta. En ella, establecemos el TAG correspondiente al periodo de tiempo en el que nos encontremos del día y fijamos el disparador que deberá ejecutarse en el siguiente periodo de tiempo según el horario. La implementación de este servicio se encuentra disponible en el Anexo D.2.2.

5.5.5. Patrones de eventos

En este apartado veremos la implementación, paso a paso, de los diferentes patrones de eventos descritos en el apartado de diseño. En primer lugar definimos el evento simple que representará la información que proviniente de los sensores. Con *Create Schema* definimos el tipo de evento *AirMeasurement* que contendrá los siguientes atributos:

```
create schema AirMeasurement
 (sensorId string, location string,
 timestamp string, pm2_5 double,
 pm10 double, o3 double, no2 double,
 so2 double, co double,
 temp double, humidity double);
```

A continuación definimos los patrones que se encargan de calcular el valor medio del contaminente correspondiente en cada sensor existente. En primer lugar le damos un nombre al patrón con *@Name('O3AndCOAvg')*, para este caso. A continuación indicamos el esquema destino con *Insert Into*. El siguiente paso es definir los atributos que vamos seleccionar con la palabra reservada *Select*. A estos atributos le podemos aplicar funciones, en este caso aplicamos la media sobre el valor de los contamiannes y, además, seleccionamos la localización, el id del sensor y el instante actual.. Indicamos el origen de los datos, con el empleo de la palabra reservada *every* le decimos al sistema que admitimos cualquier evento *AirMeasurement* y con *.win:time()* especificamos la ventana temporal de la cual queremos obtener datos. Finalmente agrupamos por el sensor.

```
@Name('O3AndCOAvg')
insert into O3AndCOAvg
select
        avg(e.o3) as o3_avg, avg(e.co) as co_avg,
        e.location as location, e.sensorId as sensorId,
        current_timestamp() as timestamp
from pattern [every e = AirMeasurement].win:time(8 hour)
group by e.sensorId;


@Name('PM10AndPM25Avg')
insert into PM10AndPM25Avg
select
        avg(e.pm10) as pm10_avg, avg(e.pm2_5) as pm2_5_avg,
        e.location as location, e.sensorId as sensorId,
```

```
        current_timestamp() as timestamp
from pattern [every e = AirMeasurement].win:time(24 hour)
group by e.sensorId;


@Name('SO2AndNO2Avg')
insert into SO2AndNO2Avg
select
        avg(e.so2) as so2_avg, avg(e.no2) as no2_avg,
        e.location as location, e.sensorId as sensorId,
        current_timestamp() as timestamp
from pattern [every e = AirMeasurement].win:time(1hour)
group by e.sensorId;
```

El siguiente paso es calcular cual es el mayor valor existente en cada localización para cada uno de los contaminentes. Al igual que en el paso anterior, definimos el nombre del patrón *@Name('O3AvgByLocation')*, seguido de *Insert Into* con el esquema destino *PollutantAvgByLocation*. A continuación, la palabra reservada *Select* con los diferentes atributos seleccionados, indicamos un id único que está compuesto por la concatenación de la localización y el instante actual, seleccionamos el nivel de alerta, la localización, el máximo de los valores para el contaminante y el instante. La fuente de datos (cláusula *From*) en este caso serán los patrones de eventos definidos anteriormente, en función del patrón deberemos indicar uno u otro. Las ventanas temporales que emplearemos serán ventanas *batch* de cinco minutos para cada contaminante. Finalmente, agrupamos con la cláusula *Group By* por localización todos los valores de contaminantes que tengan no sean nulos.

```
@Name('O3AvgByLocation')
insert into PollutantAvgByLocation
select
        e.location || current_timestamp.toString() as id,
        1 as kindPollutant, e.location as location,
        max(e.o3_avg) as val, current_timestamp() as timestamp
from pattern [every e = O3AndCOAvg].win:time_batch(5 min)
group by e.location having max(e.o3_avg) is not null;


@Name('COAvgByLocation')
insert into PollutantAvgByLocation
select
e.location || current_timestamp.toString() as id,
```

```
        2 as kindPollutant, e.location as location,
        max(e.co_avg) as val, current_timestamp() as timestamp
from pattern [every e = O3AndCOAvg].win:time_batch(5 min)
group by e.location having max(e.co_avg) is not null;


@Name('PM10AvgByLocation')
insert into PollutantAvgByLocation
select
        e.location || current_timestamp.toString() as id,
        3 as kindPollutant, e.location as location,
        max(e.pm10_avg) as val, current_timestamp() as timestamp
from pattern [every e = PM10AndPM25Avg].win:time_batch(5 min)
group by e.location having max(e.pm10_avg) is not null;


@Name('PM25AvgByLocation')
insert into PollutantAvgByLocation
select
        e.location || current_timestamp.toString() as id,
        4 as kindPollutant, e.location as location,
        max(e.pm2_5_avg) as val, current_timestamp() as timestamp
        from pattern [every e = PM10AndPM25Avg].win:time_batch(5 min)
group by e.location having max(e.pm2_5_avg) is not null;


@Name('SO2AvgByLocation')
insert into PollutantAvgByLocation
select
        e.location || current_timestamp.toString() as id,
        5 as kindPollutant, e.location as location,
        max(e.so2_avg) as val, current_timestamp() as timestamp
from pattern [every e = SO2AndNO2Avg].win:time_batch(5 min)
group by e.location having max(e.so2_avg) is not null;


@Name('NO2AvgByLocation')
insert into PollutantAvgByLocation
select
        e.location || current_timestamp.toString() as id,
```

```
        6 as kindPollutant,     e.location as location,
        max(e.no2_avg) as val,  current_timestamp() as timestamp
from pattern [every e = SO2AndNO2Avg].win:time_batch(5 min)
group by e.location having max(e.no2_avg) is not null;
```

A continuación definimos los patrones de eventos correspondientes a los diferentes niveles AQI con los valores específicos de cada contaminante. Para cada AQI, definimos su nombre *@Name("AQIL1")*, indicamos el esquema destino *AQILevels* y con la palabra *Select* determinamos los atributos (el nivel de alerta, el tipo de contaminante que lo produce, el instante, el valor del contaminante y el id). Finalmente, con la palabra reservada *From* indicamos la fuente de datos que serán los eventos definidos anteriormente (*PollutantAvgByLocation*), en esta ocasión, en la cláusula *From* indicamos también el criterio de selección que se corresponde con el rango de valores, para cada uno de los contaminantes, que produciría dicho nivel AQI descrito en cada patrón.

```
@Name("AQIL1")
insert into AQILevels
select
        1 as alertLevel, e.kindPollutant as kindPollutant,
        e.timestamp as timestamp, e.location as location,
        e.val as val, e.id as id
from pattern [every e = PollutantAvgByLocation(
                (kindPollutant = 1 and val >= 0 and val <= 0.064)
                or  (kindPollutant = 2 and val >= 0 and val <= 4.4)
                or  (kindPollutant = 3 and val >= 0 and val <= 54)
                or  (kindPollutant = 4 and val >= 0 and val <= 15.4)
                or  (kindPollutant = 5 and val >= 0 and val <= 0.034))];


@Name("AQIL2")
insert into AQILevels
select
        2 as alertLevel, e.kindPollutant as kindPollutant,
        e.timestamp as timestamp, e.location as location,
        e.val as val, e.id as id
from pattern [every e = PollutantAvgByLocation(
                (kindPollutant = 1 and val >= 0.065 and val <= 0.084)
                or  (kindPollutant = 2 and val >= 4.5 and val <= 9.4)
                or  (kindPollutant = 3 and val >= 55 and val <= 154)
                or  (kindPollutant = 4 and val >= 15.5 and val <= 40.4)
```

```
                or  (kindPollutant = 5 and val >= 0.035 and val <= 0.144))];


@Name("AQIL3")
insert into AQILevels
select
        3 as alertLevel, e.kindPollutant as kindPollutant,
        e.timestamp as timestamp, e.location as location,
        e.val as val, e.id as id
from pattern [every e = PollutantAvgByLocation(
                (kindPollutant = 1 and val >= 0.085 and val <= 0.104)
                or  (kindPollutant = 2 and val >= 9.5 and val <= 12.4)
                or  (kindPollutant = 3 and val >= 155 and val <= 254)
                or  (kindPollutant = 4 and val >= 40.5 and val <= 65.4)
                or  (kindPollutant = 5 and val >= 0.145 and val <= 0.224))];

@Name("AQIL4")
insert into AQILevels
select
        4 as alertLevel, e.kindPollutant as kindPollutant,
        e.timestamp as timestamp, e.location as location,
        e.val as val, e.id as id
from pattern [every e = PollutantAvgByLocation(
                (kindPollutant = 1 and val >= 0.105 and val <= 0.124)
                or  (kindPollutant = 2 and val >= 12.5 and val <= 15.4)
                or  (kindPollutant = 3 and val >= 255 and val <= 354)
                or  (kindPollutant = 4 and val >= 65.5 and val <= 150.4)
                or  (kindPollutant = 5 and val >= 0.225 and val <= 0.304))];


@Name("AQIL5")
insert into AQILevels
select
        5 as alertLevel, e.kindPollutant as kindPollutant,
        e.timestamp as timestamp, e.location as location,
        e.val as val, e.id as id
from pattern [every e = PollutantAvgByLocation(
                (kindPollutant = 1 and val >= 0.125 and val <= 0.374)
                or  (kindPollutant = 2 and val >= 15.4 and val <= 30.4)
                or  (kindPollutant = 3 and val >= 355 and val <= 424)
```

```
                or  (kindPollutant = 4 and val >= 150.5 and val <= 205.4)
                or  (kindPollutant = 5 and val >= 0.305 and val <= 0.604)
                or  (kindPollutant = 6 and val >= 0.65 and val <= 1.24))];


@Name("AQIL6")
insert into AQILevels
select
        6 as alertLevel, e.kindPollutant as kindPollutant,
        e.timestamp as timestamp, e.location as location,
        e.val as val, e.id as id
from pattern [every e = PollutantAvgByLocation(
                (kindPollutant = 1 and val >= 0.375)
                or  (kindPollutant = 2 and val >= 30.5)
                or  (kindPollutant = 3 and val >= 425)
                or  (kindPollutant = 4 and val >= 250.5)
                or  (kindPollutant = 5 and val >= 0.605)
                or  (kindPollutant = 6 and val >= 1.25))];
```

Finalmente será necesario agrupar estos diferentes niveles AQI que detectamos para notificar al usuario el de mayor nivel de alerta por localidad, para dicha finalidad realizamos los mismos pasos que hemos llevado a cabo anteriormente: definimos el nombre del patrón *@Name('AQIByLocation')*, seguido de la palabra *Insert Into* que especifica el esquema destino *AQIByLocation*. En la cláusula *Select* indicamos los atributos a proyectar, en este caso la concatenación de la localización junto al instante hará las veces de ID, seleccionamos además la localización, el máximo de las alertas AQI detectadas y el instante actual. En la cláusula *From*, indicamos que la fuente de datos será cualquier evento del tipo *AQILevels* en ventanas temporales de cinco segundos. Finalmente agrupamos por localización seleccionando aquellos valores de alerta no nulos.

```
@Name('AQIByLocation')
insert into AQIByLocation
select
        e.location || current_timestamp.toString() as id,
        e.location as location, max(e.alertLevel) as alertLevel,
        current_timestamp() as timestamp
from pattern [every e = AQILevels].win:time_batch(5 sec)
group by e.location having max(e.alertLevel) is not null;
```

5.6. Pruebas y validación

Durante las diferentes fases hemos ido desarrollando las partes de las que está compuesto nuestro sistema. A medida que ibamos desarrollándolo, hemos ido realizando pruebas unitarias para cada uno de los elementos que creábamos. A continuación recogemos y documentamos las pruebas unitarias y de integración que hemos realizado para verificar el correcto funcionamiento del sistema y de sus componentes.

5.6.1. Pruebas unitarias

El objetivo de las pruebas unitarias es comprobar el correcto funcionamiento de una funcionalidad en concreto. Teniendo en cuenta esto, hemos llevado a cabo las siguientes pruebas unitarias.

5.6.1.1. Detectar evento complejo en Cepheus CEP

En este caso de prueba vamos a comprobar que el motor CEP de Cepheus detecta correctamente los eventos complejos que han sido previamente definidos en el fichero de configuración.

1. Ejecutamos Orion Context Broker

   ```
   [root@system ~]# /etc/init.d/contextBroker start
   ```

2. En una terminal, accedemos a la carpeta que contiene Cepheus Broker y lo ejecutamos indicando el puerto dónde se encuentra funcionando Orion Context Broker (:1026)

   ```
   [root@system ~]# cd fiware-cepheus/cepheus-broker/
   [root@system cepheus-broker]# java -jar cepheus-broker.jar
           --remote.url="http://localhost:1026"
   ```

   ```
   2016-05-06 14:03:48.809  INFO 1728 --- [           main] com.orange.cepheus.broke
   r.Application    : Starting Application on system.novalocal with PID 1728 (/root/
   fiware-cepheus/cepheus-broker/cepheus-broker.jar started by root in /root/fiware-
   cepheus/cepheus-broker)
   2016-05-06 14:04:31.570  INFO 1728 --- [           main] com.orange.cepheus.broke
   r.Application    : Started Application in 43.161 seconds (JVM running for 43.783)
   ```

   Figura 5.18.: Cepheus Broker en funcionamiento

3. A continuación, en otra terminal, accedemos a la carpeta que contiene Cepheus CEP y lo ejecutamos con maven [6]. En caso de haber empleado el componente anteriormente, Cepheus CEP cargará automáticamente la última configuración empleada.

```
[root@system ~]# cd fiware-cepheus/cepheus-cep/
[root@system cepheus-cep]# mvn spring-boot:run
```

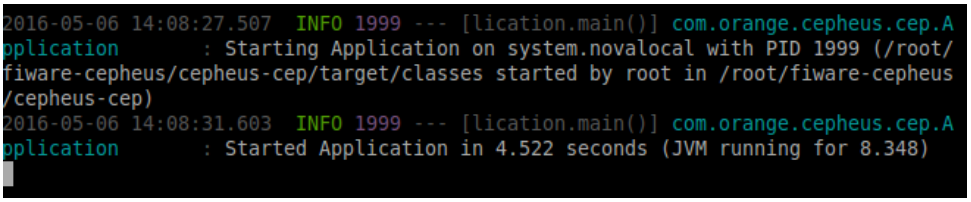

Figura 5.19.: Cepheus CEP en funcionamiento

4. Ya con el sistema en funcionamiento será necesario ejecutar el script de simulación. En este caso emplearemos un script que simula una estación para una localización, Cádiz, y envía la información necesaria para que se detecte el evento complejo de nivel 4 de calidad del aire. Para ello, empleamos el mismo fichero de configuración que hemos empleado en el desarrollo del proyecto.

```
CEP=localhost:8080
LB=localhost:8081
. ../common.sh
# Send an updateContext request with values
function sendValues() #(url, location, id, sensorId,
  temperature, humidity, pm2_5, pm10, o3, no2, so2, co) {
  payload='{
     "contextElements": [
             {
               "type": "AirMeasurement",
               "isPattern": "false",
               "id": "'$3'",
               "attributes": [
                             {
                               "name": "sensorId",
                               "type": "string",
                               "value": "'$4'"
                             },
                             {
                               "name": "location",
                               "type": "string",
                               "value": "'$2'"
                             },
                             {
```

```
      "name": "pm2_5",
      "type": "double",
      "value": '$7'
    },
    {
      "name": "pm10",
      "type": "double",
      "value": '$8'
    },
    {
      "name": "o3",
      "type": "double",
      "value": '$9'
    },
    {
      "name": "no2",
      "type": "double",
      "value": '${10}'
    },
    {
      "name": "so2",
      "type": "double",
      "value": '${11}'
    },
    {
      "name": "co",
      "type": "double",
      "value": '${12}'
    },
    {
      "name": "temperature",
      "type": "double",
      "value": '$5'
    },
    {
      "name": "humidity",
      "type": "double",
      "value": '$6'
    }
  ]
```

```
            }
          ],
          "updateAction": "APPEND"
    }'
        send $1 "v1/updateContext" "$payload"
}

echo "Example with 1 sensor in Cadiz that produces level 4 for AQI"

echo "#1 First update CEP with AirMeasurement configuration and
                Complex Events"
CONFIG=`cat config.json`
updateConfig $CEP "$CONFIG"

echo ""
echo "#2 Sending values..."
echo ""

sleep 10
out=$(sendValues $LB "Cadiz" "AirMeasurementCadiz1" "sensor001"
        "15" "75" "300" "0" "0" "0" "0" "0")
```

5. Ejecutamos el script.

```
[root@system system]# sh runPruebaCEP.sh
```

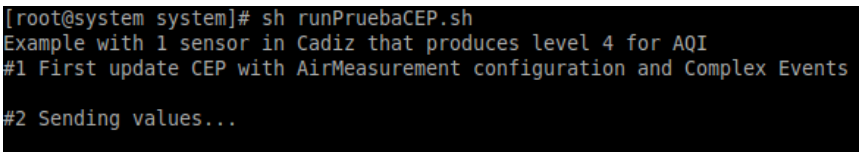

Figura 5.20.: Ejecución del script

6. Como podemos observar en la figura 5.21, en primer lugar Cepheus agrega la configuración (si no estaba), a su configuración de ejecución, es decir agrega los eventos simples que recibirá, los eventos complejos que producirá y los patrones de eventos.

7. A continuación se envia un registro que simula al sensor «sensor001» ubicado en Cadiz, como podemos observar en la figura 5.22.

8. Los valores de este registro provoca varios eventos complejos de los que hemos

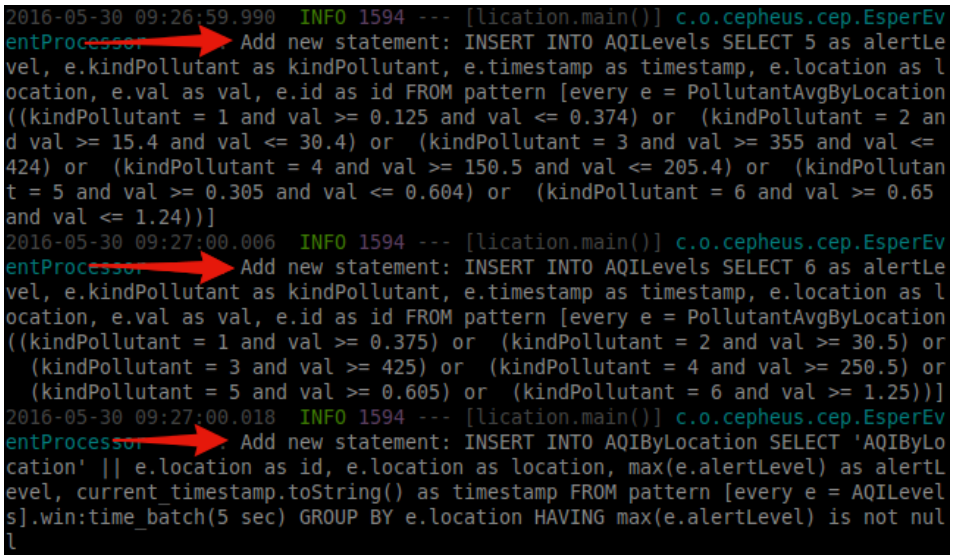

Figura 5.21.: Agregando elementos configuración

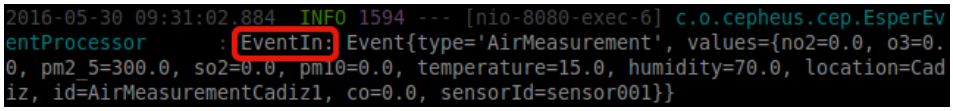

Figura 5.22.: Evento simple de entrada

definidos, pero el que nos interesa es el de nivel de calidad 4 debido al valor del contaminante PM 10, como podemos observar en la figura 5.23.

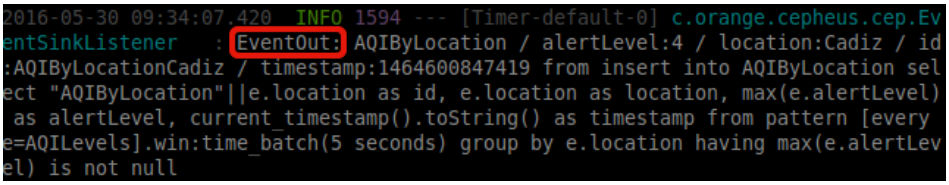

Figura 5.23.: Evento complejo *AQIByLocation*

5.6.1.2. Realizar suscripción a Orion Context Broker

En este caso de pruebas vamos a comprobar que la suscripción a eventos complejos de un determinado tipo se realiza correctamente.

1. Ejecutamos Orion Context Broker

   ```
   [root@system ~]# /etc/init.d/contextBroker start
   ```

2. A continuación iniciamos nuestro consumidor para el evento complejo que deseemos gestionar. En este caso, ejecutaremos el que se encarga de detectar los eventos complejos relativos a los niveles de calidad de aire.

   ```
   [root@system system]# ./accumulator-server.py 1028 /accumulate on
   ```

3. Realizamos la suscripción a los eventos complejos que deseemos, en este caso los relativos a la calidad del aire. Para ello, en otra terminal, ejecutaremos el siguiente comando:

   ```
   (curl localhost:1026/v1/subscribeContext -s -S --header 'Content-Type:
   application/json' \
   --header 'Accept: application/json' -d @- | python -mjson.tool)
   <<EOF
   {
       "entities": [
           {
               "type": "AQIByLocation",
               "isPattern": "true",
               "id": "AQIByLocation.*"
           }
       ],
       "attributes": [
           "alertLevel",
   ```

```
            "location",
            "timestamp"
        ],
        "reference": "http://localhost:1028/accumulate",
        "duration": "PT1H",
        "notifyConditions": [
            {
                "type": "ONCHANGE",
                "condValues": [
                    "alertLevel"
                ]
            }
        ]
    }
    EOF
```

4. En la misma terminal que hemos ejecutado el comando de suscripción aparecerá la respuesta por parte de Orion Context Broker, figura 5.24, indicando la duración y el ID de la suscripción.

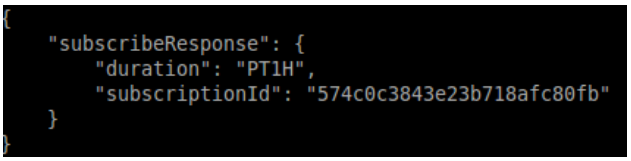

Figura 5.24.: Respuesta suscripción a Orion Context Broker

5.6.1.3. Notificación de evento complejo en el consumidor

En este caso de prueba vamos a comprobar que, una vez detectado un evento complejo determinado, este sea notificado al consumidor que previamente se ha suscrito a dicho evento complejo detectado.

1. Para la realización de esta prueba debemos tener:

   a) Orion Context Broker en funcionamiento.

   b) Cepheus CEP y Cepheus Broker en funcionamiento.

   c) Consumidor en funcionamiento y con la suscripción activada a los eventos complejos de nivel de calidad del aire.

2. Por lo tanto, ejecutaremos el mismo script que utilizamos en la prueba de detección de eventos complejos, ya que como probamos anteriormente, se detecta un evento complejo relativo a nivel 4 de calidad del aire.

   ```
   [root@system system]# sh runPruebaCEP.sh
   ```

3. Al igual que antes se enviará la configuración al CEP, si no está fijada se establecerá y enviaremos el registro que simula al «sensor001» de Cádiz. Se registrará el evento de entrada y los correspondientes eventos complejos detectados. En este caso el que nos interesa es "AQIByLocation" , figura 5.25, ya que en nuestra condición de suscripción nos suscribimos a los eventos de este tipo.

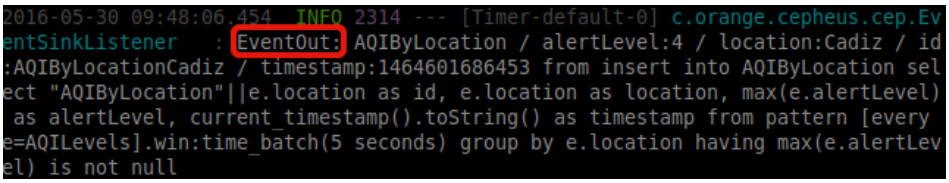

Figura 5.25.: Evento complejo *AirQualityAlert*

4. Y en la terminal del consumidor veremos, figura 5.26, que hemos detectado el evento complejo con id "AQIByLocationCadiz" que corresponde al evento complejo de nivel 4 de calidad del aire en Cádiz.

5.6.1.4. Notificación de alerta de calidad del aire

En este caso de prueba vamos a comprobar que, una vez detectado un evento complejo determinado y notificado al consumidor, este invoca correctamente al Servicio Web que se encarga de crear la notificación.

1. Para la realización de esta prueba debemos tener:

   a) Orion Context Broker en funcionamiento.

   b) Cepheus CEP y Cepheus Broker en funcionamiento.

   c) Consumidor en funcionamiento y con la suscripción activada a los eventos complejos de nivel de calidad del aire.

   d) Servidor LAMPP en funcionamiento.

   ```
   [root@system ~]# /opt/lampp/lampp start
   ```

   e) La aplicación Android instalada en un dispositivo.

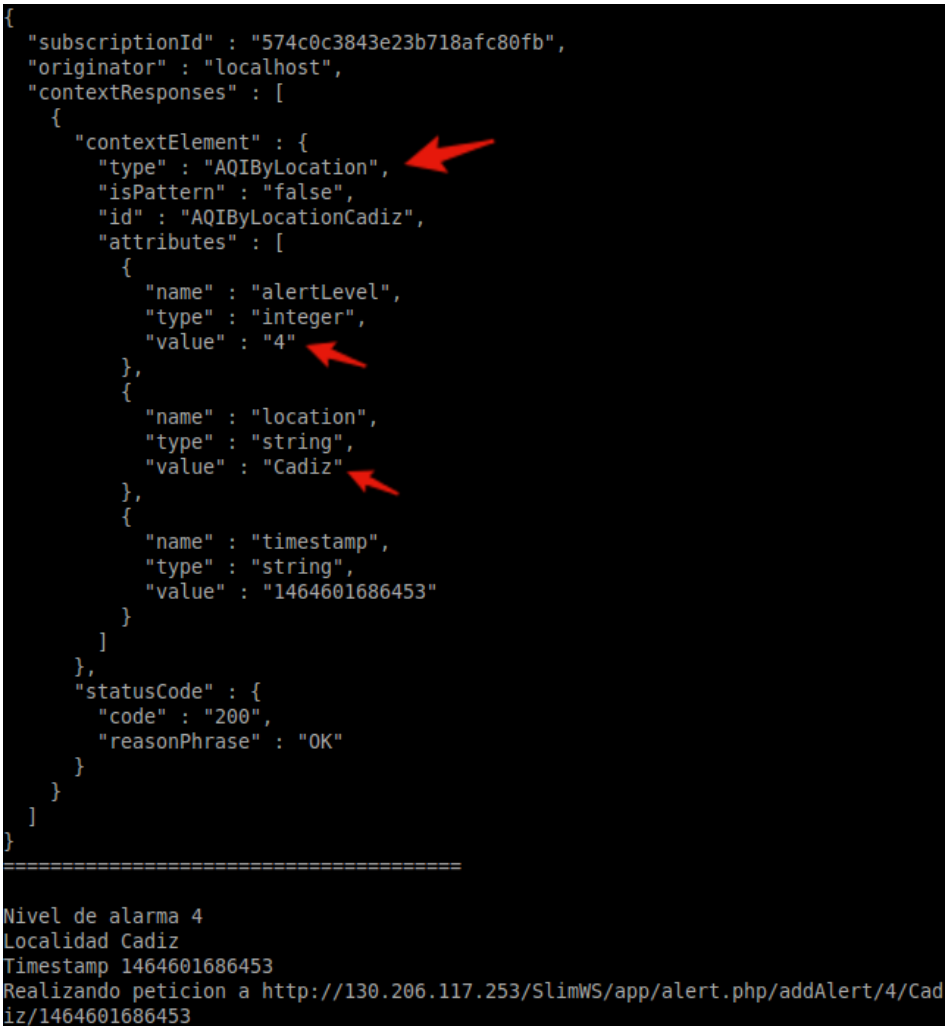

Figura 5.26.: Evento complejo notificado en el consumidor

2. Se da el caso de que detectamos un evento complejo, por ejemplo, de alerta de calidad del aire. Como podemos observar en la figura 5.27, este invoca al Servicio Web de contenido en la url http://130.206.117.253/SlimWS/app/alert.php/addAlert/4/Cadiz/1462704599818 que contiene los parámetros relativos a la alerta que acabamos de detectar.

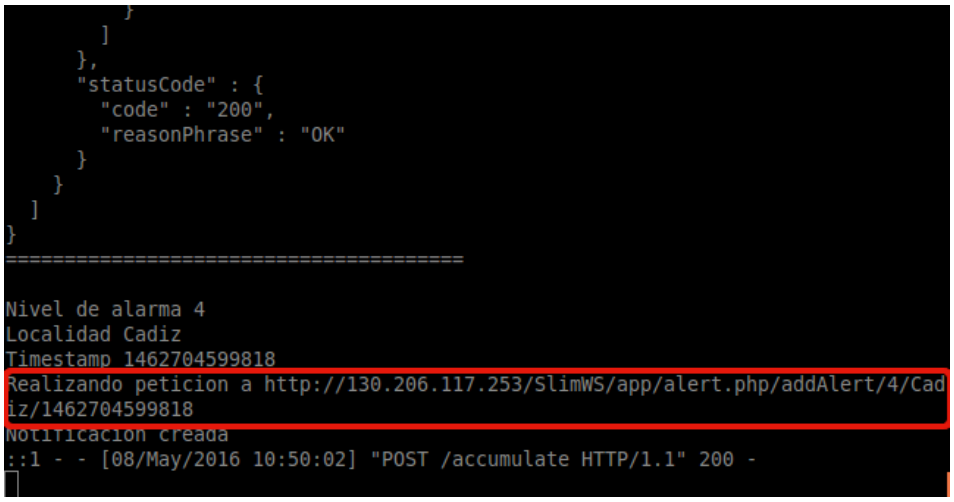

Figura 5.27.: Invocación Servicio Web desde el consumidor

3. De igual forma podemos acceder directamente al enlace para probar el funcionamiento del Servicio Web. Al acceder a ese enlace, ejecutaremos el Servicio Web y visualizaremos que la alerta ha sido creada correctamente, figura 5.28, y el JSON que genera la API de PushBots al enviar la notificación. Por otro lado, en nuestro teléfono visualizaremos la notificación de la alerta, figuras 5.29 y 5.30.

5.6.1.5. Monitorización del contexto mediante GPS

En este caso de prueba vamos a comprobar que, haciendo uso de la monitorización con el sensor GPS del dispositivo móvil, la monitorización del contexto del usuario se realiza correctamente. Para ello, vamos a mostrar capturas de diferentes momentos, una estático y otra caminando, y su correspondiente TAG aplicado en el sistema de notificaciones PushBots.

1. Usuario estático en Cádiz.

   a) En este caso, el usuario se encuentra en una posición estática, es decir no realiza desplazamientos y sus coordenadas coinciden con la localización Cádiz, figuras 5.31 y 5.32.

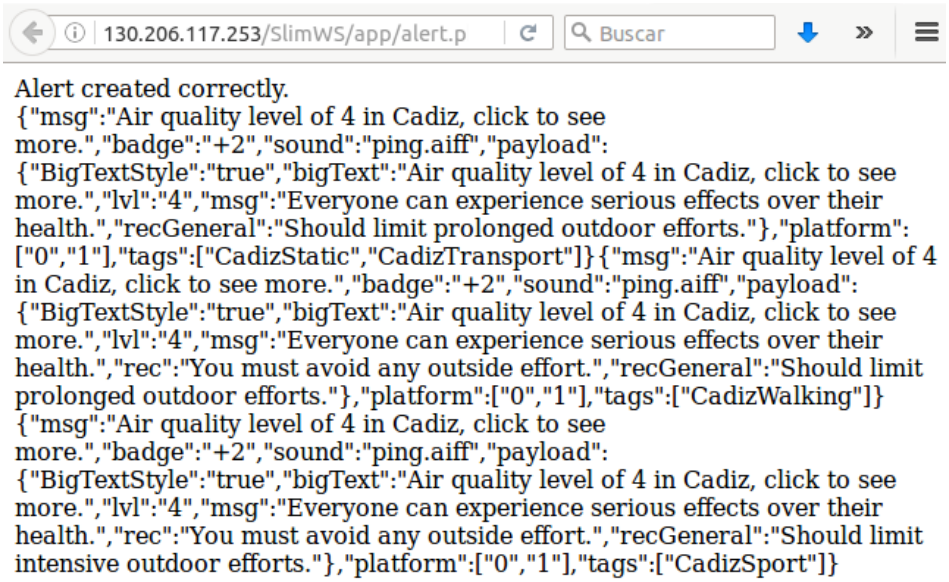

Figura 5.28.: Respuesta del Servicio Web en el navegador

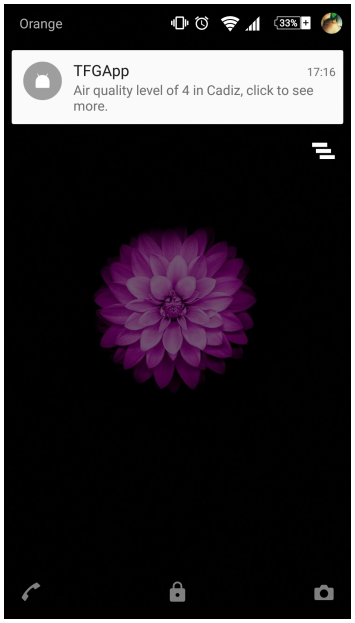

Figura 5.29.: Globo de notificación

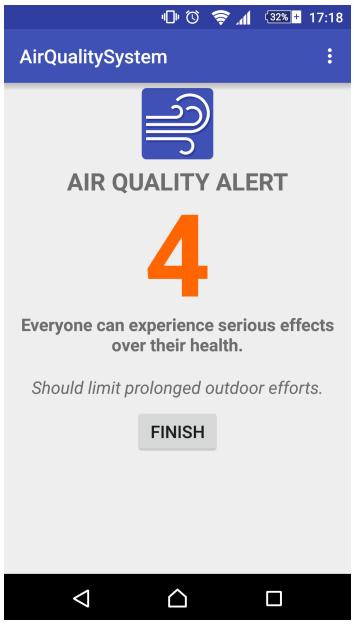

Figura 5.30.: Notificación en la aplicación

Figura 5.31.: Usuario estático en Cádiz en Android

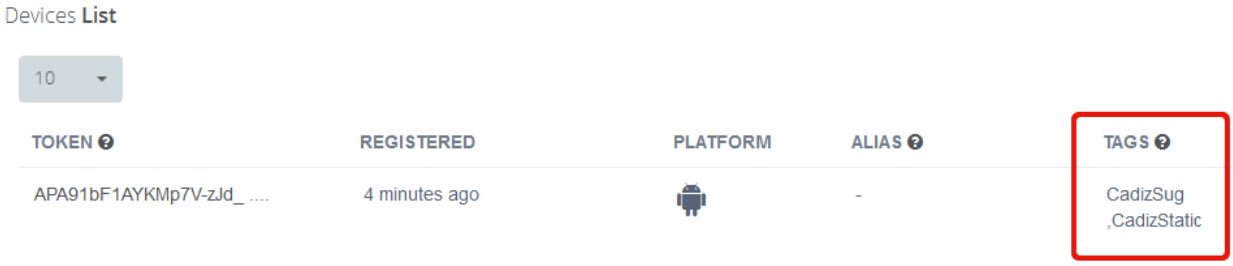

Figura 5.32.: TAG *CadizStatic* en PushBots

2. Usuario caminando en Cádiz.

   a) En este caso, el usuario se encuentra caminando a una velocidad de unos 1.5 metros por segundo (5,3 km/h), figuras 5.33 y 5.34.

Figura 5.33.: Usuario caminando en Cádiz en Android

5.6.1.6. Monitorización del contexto mediante horario

En este caso de prueba vamos a comprobar que, haciendo uso de la monitorización por horario/hábitos, previamente establecidos, el TAG del usuario se irá actualizando al acorde en cada instante.

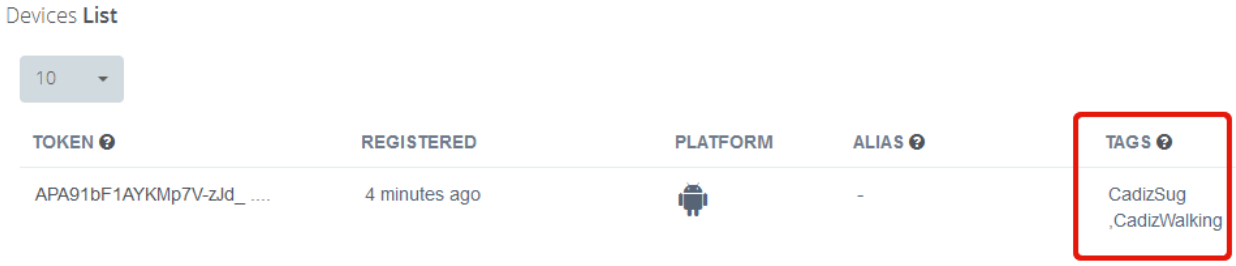

Figura 5.34.: TAG *CadizWalking* en PushBots

1. En primer lugar definimos el horario para el día actual, en nuestro caso el Lunes, como vemos en la figura 5.35.

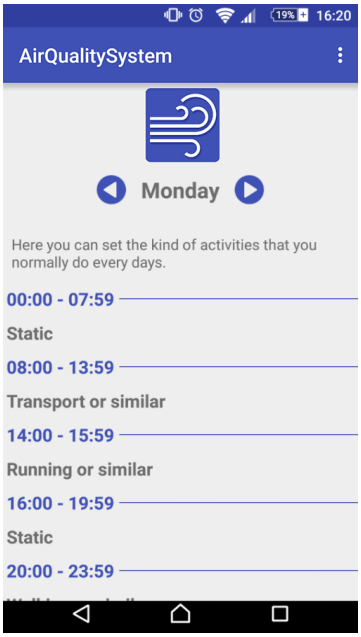

Figura 5.35.: Horario para el día lunes

2. A continuación seleccionamos la opción "*Timetable*" en la pantalla principal de la aplicación para comenzar la monitorización por horario. Como podemos apreciar en la figura 5.36, la actividad actual sería estático, luego el TAG sería *CadizStatic* como podemos apreciar en la figura 5.37.

3. A continuación, en las opciones del términal Android, vamos a modificar la hora y vamos a pasar al inicio siguiente fragmento horario, que sería el comprendido entre las 20:00 y las 23:59. Luego adelantamos el reloj a las 20:00

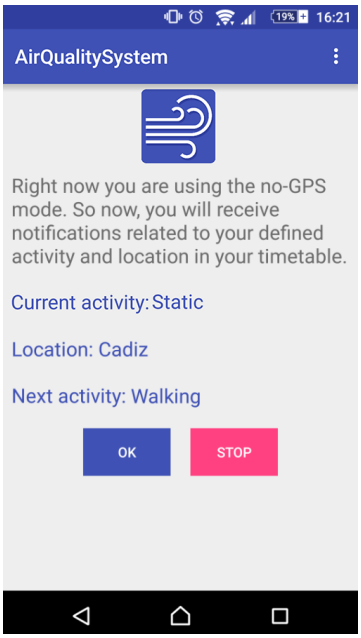

Figura 5.36.: Monitorización por horario a las 16:21h

Figura 5.37.: TAG *CadizStatic* en PushBots

y volvemos a la pantalla de monitorización por horario, figura 5.38. Ahora el TAG ha cambiado ya que hemos entrado en el siguiente periodo de tiempo, cuya actividad está relacionada con caminar, luego el TAG será *CadizWalking* tal y como podemos apreciar en la figura 5.39.

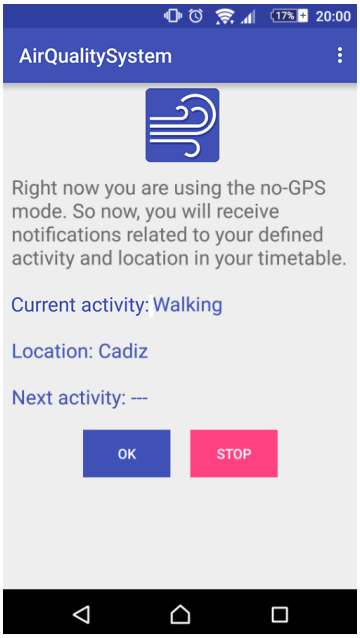

Figura 5.38.: Monitorización por horario a las 20:00h

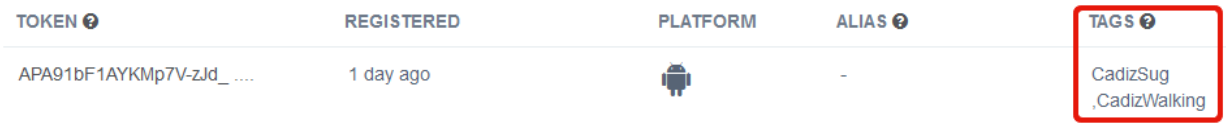

Figura 5.39.: TAG *CadizTransport* en PushBots

5.6.1.7. Consultar alertas de calidad del aire detectadas

En este caso de prueba vamos a comprobar que se detectan las alertas de calidad del aire de una localización dada desde la aplicación Android.

1. En primer lugar debemos tener el sistema en funcionamiento y el servidor LAMPP en ejecución.

2. A continuación ejecutamos la aplicación y accedemos al menú "*Alers detected*". En este fragmento visualizaremos las distintas alertas para la ubicación desde la cual estamos haciendo la petición como podemos ver en la figura 5.40.

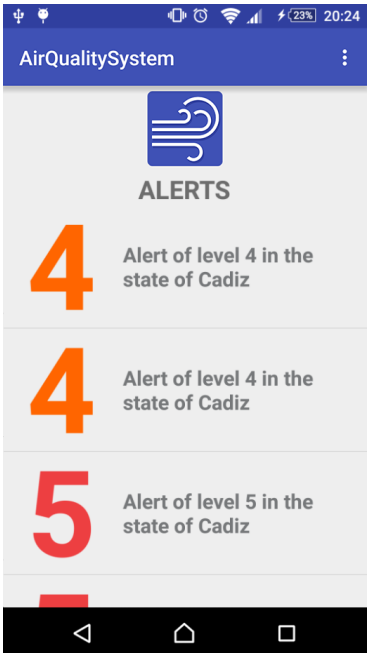

Figura 5.40.: Alertas detectadas en Cádiz

3. Por otro lado, si accedemos al siguiente enlace: http://130.206.117.253/SlimWS/app/getAlertsByLocation.php/alerts/Cadiz podemos ver qué es lo que devuelve el Servicio Web que la propia aplicación invoca en el apartado de alertas detectadas, figura 5.41.

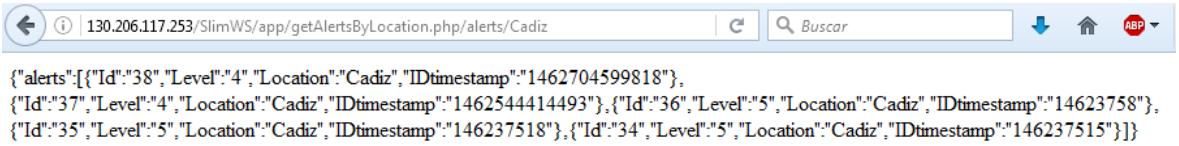

Figura 5.41.: Invocación Servicio Web para consultar alertas en Cádiz

5.6.1.8. Patrones de eventos complejos

En este caso de prueba vamos a comprobar, mediante el compilador online de Esper, que el funcionamiento de los patrones complejos a detectar es el esperado, es decir, están bien definidos y su comportamiento es el adecuado.

1. En primer lugar accedemos al compilador online de Esper: http://esper-epl-tryout.appspot.com/epltryout/index.html, aceptamos los términos de uso pulsando en el botón azul "*Enter*".

2. A continuación encontramos una disposición en tres ventanas. En la ventana de la izquierda escribiremos las definiciones de las diferentes sentencias EPL. En la ventana central introduciremos el flujo de datos, es decir, los eventos simples que mandamos al compilador. Finalmente, en la ventana de la derecha veremos los resultados de la simulación.

3. Por lo tanto, en la ventana de la izquierda colocamos los mismos patrones de eventos complejos que hemos definido en la sección 5.4.1.1

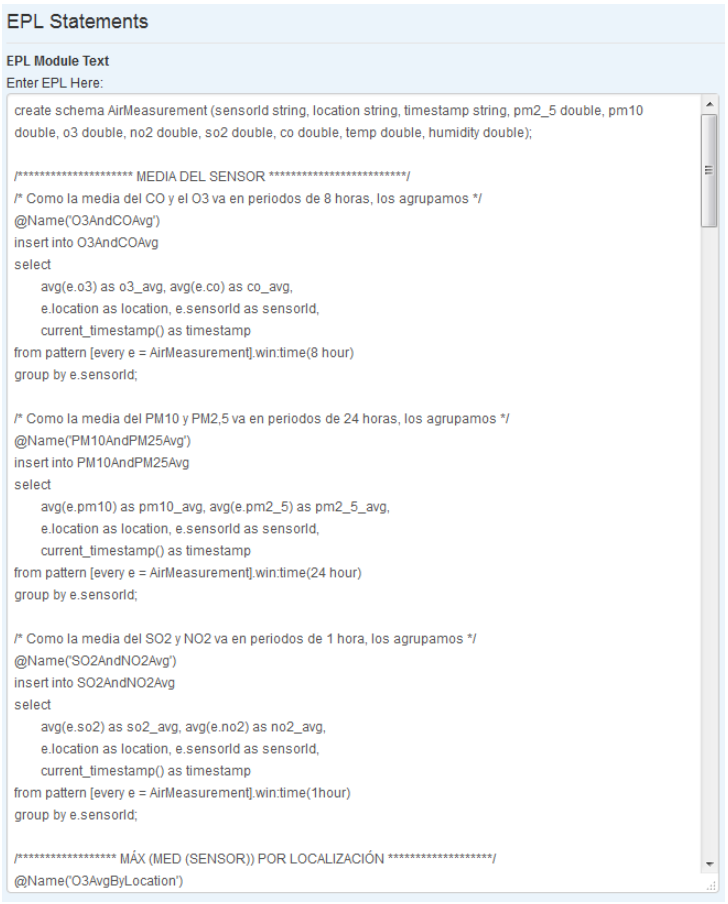

Figura 5.42.: Sentencias EPL definidas

4. En la ventana central introduciremos el siguiente flujo de datos. Este flujo de datos contiene comentarios para poder identificar, en todo momento, que es lo que se detecta y por qué en cada instate.

```
/* AQI L1 - Cadiz y Sevilla*/
AirMeasurement={sensorId='Cadiz001',location='Cadiz',timestamp='1',
```

```
  o3=0,co=0,pm10=0,pm2_5=0,so2=0,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Cadiz002',location='Cadiz',timestamp='2',
  o3=0.05,co=0,pm10=0,pm2_5=5,so2=0,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Sevilla001',location='Sevilla',timestamp='3',
  o3=0,co=0,pm10=0,pm2_5=0,so2=0,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Sevilla002',location='Sevilla',timestamp='4',
  o3=0.05,co=0,pm10=0,pm2_5=5,so2=0,no2=0,temp=10,humidity=50}

/* Media de todos los contaminantes = 0 => Nivel de alerta 1*/
t=t.plus(6 min);


/* AQI L2 - Cádiz  /  AQI L1 - Sevilla*/
AirMeasurement={sensorId='Cadiz001',location='Cadiz',timestamp='5',
  o3=0,co=0,pm10=150,pm2_5=0,so2=0.04,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Cadiz002',location='Cadiz',timestamp='6',
  o3=0.05,co=12,pm10=0,pm2_5=5,so2=0.03,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Sevilla001',location='Sevilla',timestamp='7',
  o3=0,co=0,pm10=30,pm2_5=10,so2=0,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Sevilla002',location='Sevilla',timestamp='8',
  o3=0.05,co=0,pm10=20,pm2_5=5,so2=0,no2=0,temp=10,humidity=50}

/*
En este caso para Cádiz, el contaminante PM10 presenta un valor medio
de 75, luego se detecta una alerta de calidad de nivel 2.

En este caso para Sevilla, ninguna media de contaminante sobrepasa
el nivel de alerta 1, luego permanece igual
*/
t=t.plus(6 min);

/* AQI L2 - Cádiz  /  AQI L3 - Sevilla */
AirMeasurement={sensorId='Cadiz001',location='Cadiz',timestamp='9',
  o3=0,co=0,pm10=150,pm2_5=0,so2=0.04,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Cadiz002',location='Cadiz',timestamp='10',
  o3=0.05,co=12,pm10=0,pm2_5=5,so2=0.03,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Sevilla001',location='Sevilla',timestamp='11',
  o3=0.147,co=0,pm10=300,pm2_5=10,so2=0,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Sevilla002',location='Sevilla',timestamp='12',
  o3=0.175,co=0,pm10=250,pm2_5=5,so2=0.4,no2=0,temp=10,humidity=50}
```

```
/*
En este caso para Cádiz, el valor de los contaminantes se mantiene
luego el nivel de alerta 2 de calidad no varía.

En este caso para Sevilla, el valor del PM10 se dispara, por lo
tanto se detecta una alerta de calidad de nivel 3.
*/
t=t.plus(6 min);

/* AQI L4 - Cádiz  /  AQI L4 - Sevilla */
AirMeasurement={sensorId='Cadiz001',location='Cadiz',timestamp='9',
  o3=0.275,co=0,pm10=700,pm2_5=450,so2=0.04,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Cadiz002',location='Cadiz',timestamp='10',
  o3=0.05,co=12,pm10=650,pm2_5=5,so2=1,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Sevilla001',location='Sevilla',timestamp='11',
  o3=0.147,co=38,pm10=750,pm2_5=10,so2=0,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Sevilla002',location='Sevilla',timestamp='12',
  o3=0.222,co=0,pm10=250,pm2_5=5,so2=0.7,no2=0,temp=10,humidity=50}

/*
En este caso para Cádiz, el valor del PM10 se dispara, por lo
tanto se detecta una alerta de calidad de nivel 3.

En este caso para Sevilla, el valor del PM10 sigue subiendo,
por lo tanto su valor medio y originando una alerta de nivel 4.
*/
t=t.plus(6 min);

/* AQI L5 - Cádiz  /  AQI L3 - Sevilla */
AirMeasurement={sensorId='Cadiz001',location='Cadiz',timestamp='13',
  o3=0.5,co=0,pm10=700,pm2_5=450,so2=0.04,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Cadiz002',location='Cadiz',timestamp='14',
  o3=0.05,co=12,pm10=650,pm2_5=5,so2=1,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Sevilla001',location='Sevilla',timestamp='15',
  o3=0,co=0,pm10=0,pm2_5=0,so2=0,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Sevilla002',location='Sevilla',timestamp='16',
  o3=0,co=0,pm10=0,pm2_5=0,so2=0,no2=0,temp=10,humidity=50}

/*
```

```
En este caso para Cádiz, el valor del O3, PM10 y PM2,5 se disparan,
dando lugar, cada uno de ellos, a alertas de nivel de calidad 5.

En este caso para Sevilla, actualizamos los valores de los contaminantes,
a 0, bajando sus valores medios pero manteniendo el valor del O3 y SO2,
suficientemente altos como para detectar alerta de nivel 3.
*/

/*
ADELANTAMOS EL TIEMPO 24H, DE TAL FORMA QUE TODAS LAS VENTANAS
TEMPORALES DE CONTAMINANTES SE HAN ''REINICIADO''. POR LO TANTO
EL VALOR QUE PONGAMOS PARA EL CONTAMINANTE, AL SER EL PRIMERO,
SERÁ EL QUE DETERMINE DIRECTAMENTE EL NIVEL DE ALERTA AQI
*/

t=t.plus(24 hours);

/* AQI L6 Cadiz y AQI L6 Sevilla */
AirMeasurement={sensorId='Cadiz001',location='Cadiz',timestamp='17',
  o3=0.01,co=0,pm10=700,pm2_5=450,so2=0.04,no2=0,temp=10,humidity=50}
AirMeasurement={sensorId='Sevilla001',location='Sevilla',timestamp='18',
  o3=0.4,co=0,pm10=0,pm2_5=125,so2=0,no2=0,temp=10,humidity=50}


/*
En este caso para Cádiz, el valor del PM10 y PM2,5 originan alertas
de nivel de calidad 6.

Finalmente, para Sevilla, el valor del Ozono da lugar a una alerta
de nivel de calidad 6 también.
*/
t=t.plus(6 min)
```

5. Finalmente, pulsamos en el botón "*Submit*" que está justo encima de la ventana central, el resultado será similar al de la figura 5.43.

5.6.2. Pruebas de integración

Una vez realizadas las pruebas unitarias, hemos llevado a cabo la siguiente prueba de integración para verificar que la comunicación y el funcionamiento de los componentes de forma conjunta es correcto.

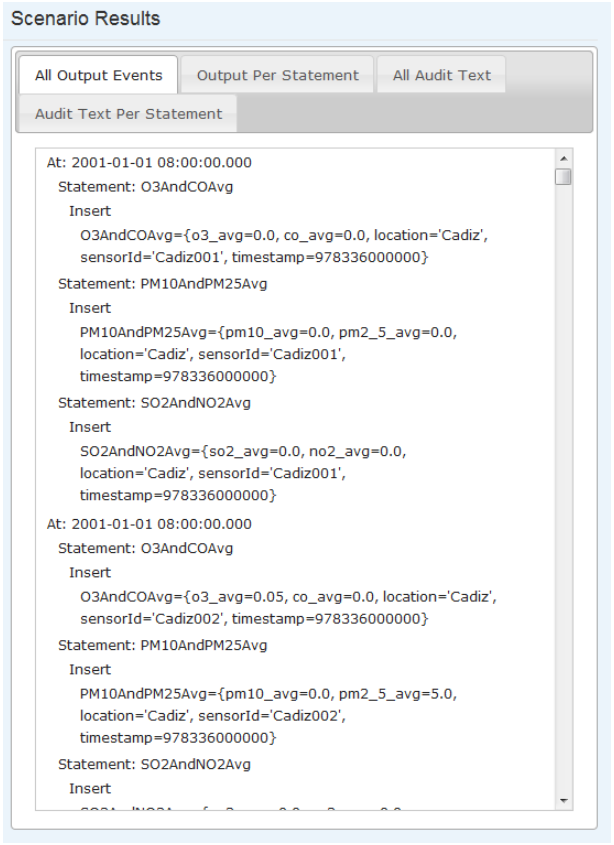

Figura 5.43.: Resultado simulación

5.6.2.1. Prueba del sistema completo

Puesto que el sistema está dividido en dos partes fundamentales: FIWARE y Android, y que hemos probado el correcto funcionamiento de estas partes y sus módulos de forma individual, a continuación realizaremos una prueba del sistema en completo funcionamiento.

Dicha prueba consistirá en la ejecución de un script que simulará una serie de alertas de calidad en unos sensores ubicados en Cádiz y en Sevilla. Para agilizar la prueba, las ventanas temporales de 5 minutos con las que hemos definido los patrones de eventos las vamos a reducir a 30 segundos que coincide con el tiempo que pasará entre lote de eventos enviados por el simulador. La prueba consistirá en dos sensores por ciudad, «Cadiz001», «Cadiz002», «Sevilla001» y «Sevilla002» que enviarán los siguientes datos:

```
### Iteración 1 ###
Cadiz001 | O3 = 0.064; CO = 2.2; PM10 = 10; PM2,5 = 2; SO2 = 0.001; NO2 = 0;
Cadiz002 | O3 = 0.032; CO = 1.1; PM10 = 35; PM2,5 = 12; SO2 = 0.005; NO2 = 0;
Sevilla001 | O3 = 0.054; CO = 2.4; PM10 = 15; PM2,5 = 7; SO2 = 0.007; NO2 = 0;
Sevilla002 | O3 = 0.012; CO = 0.8; PM10 = 10; PM2,5 = 13; SO2 = 0.012; NO2 = 0;

Dará lugar a alerta de nivel 1 en Cádiz y en Sevilla.

### Iteración 2 ###
Cadiz001 | O3 = 0.084; CO = 3.4; PM10 = 40; PM2,5 = 10; SO2 = 0.025; NO2 = 0.2;
Cadiz002 | O3 = 0.054; CO = 2.5; PM10 = 55; PM2,5 = 15; SO2 = 0.04; NO2 = 0.1;
Sevilla001 | O3 = 0.04; CO = 3.5; PM10 = 50; PM2,5 = 13; SO2 = 0.04; NO2 = 0.1;
Sevilla002 | O3 = 0.05; CO = 6; PM10 = 50; PM2,5 = 6; SO2 = 0.012; NO2 = 0.17;

Dará lugar a alerta de nivel 2 en Cádiz y nivel 1 en Sevilla.

### Iteración 3 ###
Cadiz001 | O3 = 0.02; CO = 4; PM10 = 50; PM2,5 = 7; SO2 = 0.03; NO2 = 0.2;
Cadiz002 | O3 = 0.03; CO = 5; PM10 = 0; PM2,5 = 5; SO2 = 0.03; NO2 = 0;
Sevilla001 | O3 = 0.12; CO = 3; PM10 = 25; PM2,5 = 12; SO2 = 0.04; NO2 = 0.14;
Sevilla002 | O3 = 0.02; CO = 5.5; PM10 = 100; PM2,5 = 12; SO2 = 0.04; NO2 = 0.15;

Dará lugar a alerta de nivel 1 en Cádiz y nivel 2 en Sevilla.

### Iteración 4 ###
Cadiz001 | O3 = 0.07; CO = 5; PM10 = 300; PM2,5 = 50; SO2 = 0.15; NO2 = 0.24;
Cadiz002 | O3 = 0.3; CO = 12; PM10 = 150; PM2,5 = 50; SO2 = 0.14; NO2 = 0.5;
```

```
Sevilla001 | O3 = 0.2; CO = 14; PM10 = 25; PM2,5 = 47; SO2 = 0.14; NO2 = 0.1;
Sevilla002 | O3 = 0; CO = 12; PM10 = 500; PM2,5 = 14; SO2 = 0.05; NO2 = 0;

Dará lugar a alerta de nivel 3 en Cádiz y en Sevilla.

### Iteración 5 ###
Cadiz001 | O3 = 0; CO = 0; PM10 = 0; PM2,5 = 0; SO2 = 0; NO2 = 0;
Cadiz002 | O3 = 0; CO = 0; PM10 = 0; PM2,5 = 0; SO2 = 0; NO2 = 0;
Sevilla001 | O3 = 0; CO = 0; PM10 = 0; PM2,5 = 0; SO2 = 0; NO2 = 0;
Sevilla002 | O3 = 0; CO = 0; PM10 = 0; PM2,5 = 0; SO2 = 0; NO2 = 0;

Dará lugar a alerta de nivel 2 en Cádiz y en Sevilla.
```

1. En primer lugar debemos arrancar los diferentes componentes del sistema: Orion, Cepheus CEP y Cepheus Broker. Es recomendable el uso del software Terminator para poder operar en diferentes terminales vía SHH de forma compacta y eficiente, figura 5.44.

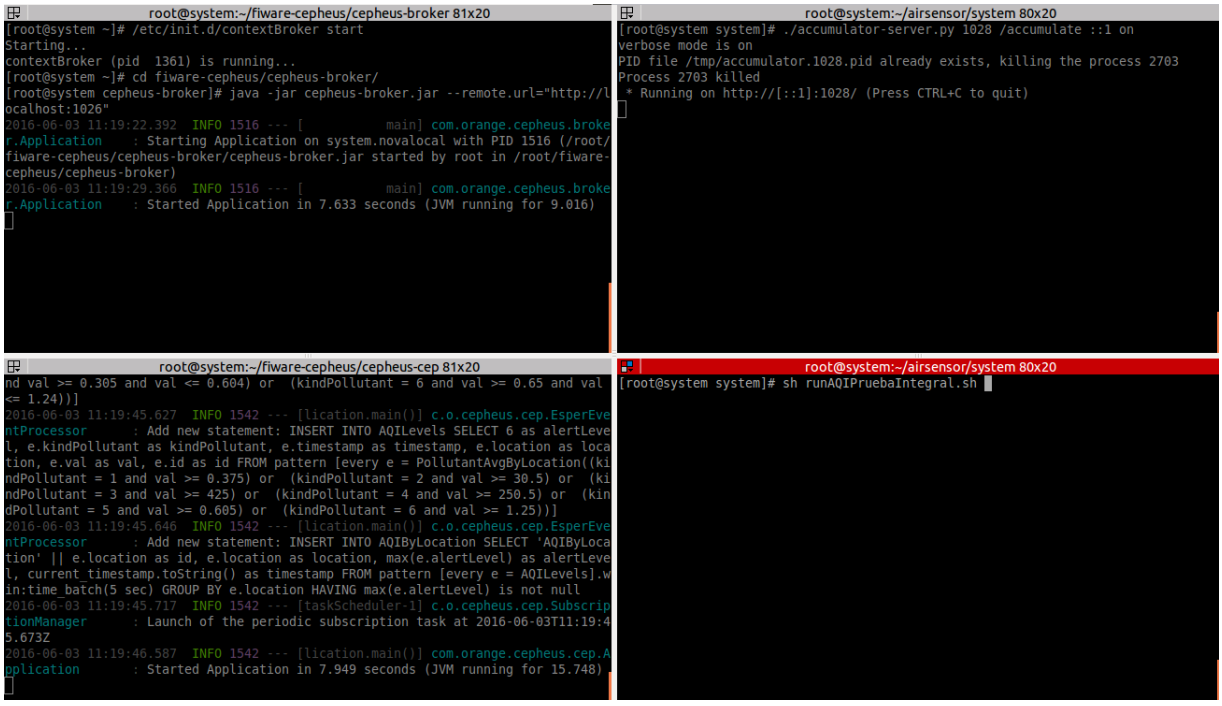

Figura 5.44.: Software Terminator con 4 terminales activas

a) En una primera terminal iniciamos Orion y Cepheus Broker.

```
[root@system ~]# /etc/init.d/contextBroker start
[root@system ~]# cd fiware-cepheus/cepheus-broker/
[root@system cepheus-broker]# java -jar cepheus-broker.jar
        --remote.url="http://localhost:1026"
```

b) En una segunda terminal ejecutamos Cepheus CEP.

```
[root@system ~]# cd fiware-cepheus/cepheus-ce/
[root@system cepheus-cep]# mvn spring-boot:run
```

2. A continuación iniciaremos el consumidor de eventos, para ello, en una tercera terminal ejecutamos los siguientes comandos.

```
[root@system ~]# cd airsensor/system/
[root@system system]# ./accumulator-server.py 1028 /accumulate on
```

3. En este momento tendremos tres terminales en funcionamiento, una con Cepheus Broker, otra con Cepheus CEP y finalmente otra con el consumidor de eventos. Abrimos una cuarta y definitiva.

a) En esta cuarta terminal iniciamos el servidor LAMPP.

```
[root@system ~]# /opt/lampp/lampp start
```

b) A continuación enviamos la condición de suscripción a los eventos complejos relativos a las alerta de calidad del aire agrupadas por localización. Este comando nos devolverá nuestro ID de suscripción. Finalmente, podemos iniciar la ejecución. Para ello haremos uso del script runPruebaIntegral.sh cuyo código simula el flujo de datos que previamente hemos definido.

```
(curl localhost:1026/v1/subscribeContext -s -S --header 'Content-Type:
application/json' \
--header 'Accept: application/json' -d @- | python -mjson.tool) <<EOF
{
    "entities": [
        {
            "type": "AQIByLocation",
            "isPattern": "true",
            "id": "AQIByLocation.*"
        }
    ],
    "attributes": [
        "alertLevel",
        "location",
        "timestamp"
```

```
        ],
        "reference": "http://localhost:1028/accumulate",
        "duration": "PT1H",
        "notifyConditions": [
            {
                "type": "ONCHANGE",
                "condValues": [
                    "alertLevel"
                ]
            }
        ]
    }
    EOF
```

4. A partir de este momento nuestro sistema empezará a recibir los eventos y comenzará su cómputo para la detección de eventos complejos previamente definidos en el fichero de configuración. En las figuras 5.45 y 5.46 podemos observar los diferentes eventos complejos de alerta 1 y 3 detectados en la terminal de Cepheus CEP. En las figuras 5.47 y 5.48 podemos ver los eventos anteriormente citados, en este caso para la localidad de Cádiz, notificados y procesados en el consumidor. Finalmente, en las figuras 5.49 y 5.50 podemos ver dos de las alertas que hemos recibido durante este proceso en el terminal Android.

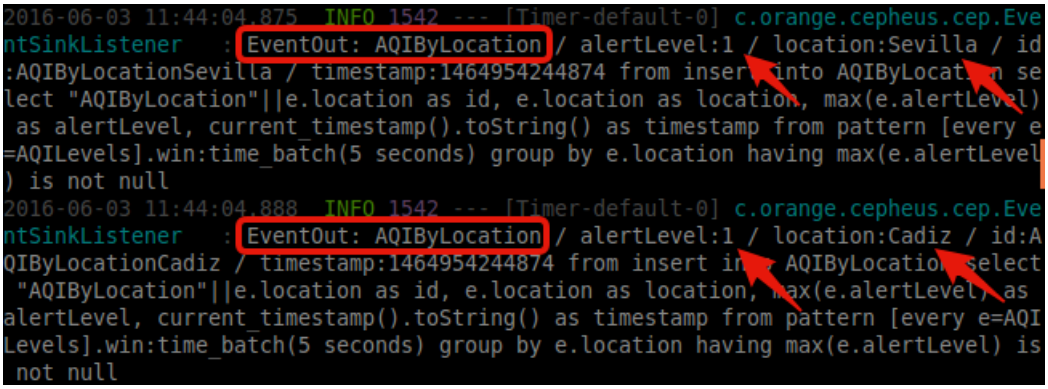

Figura 5.45.: Alerta de nivel 1 en Cádiz y Sevilla Terminal Cepheus CEP

5. De esta forma queda testado el sistema de forma completo, asegurando su correcto funcionamiento de los módulos trabajando de forma conjunta.

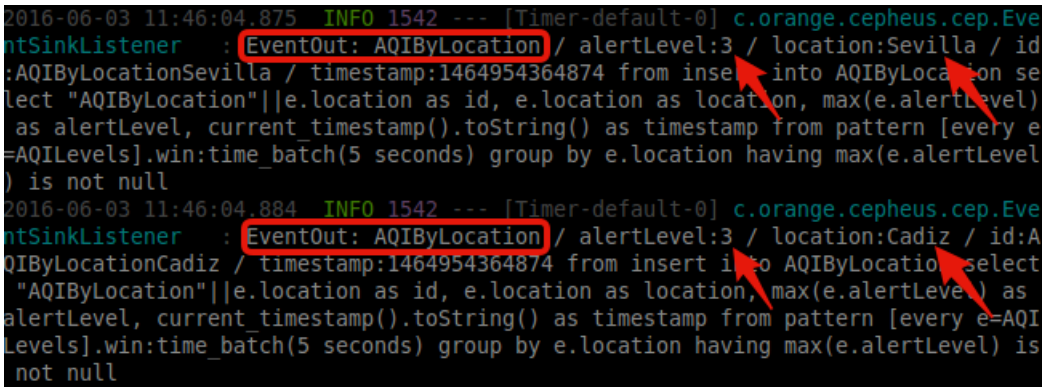

Figura 5.46.: Alerta de nivel 3 en Cádiz y Sevilla Terminal Cepheus CEP

Figura 5.47.: Alerta de nivel 1 en Cádiz Terminal Consumidor

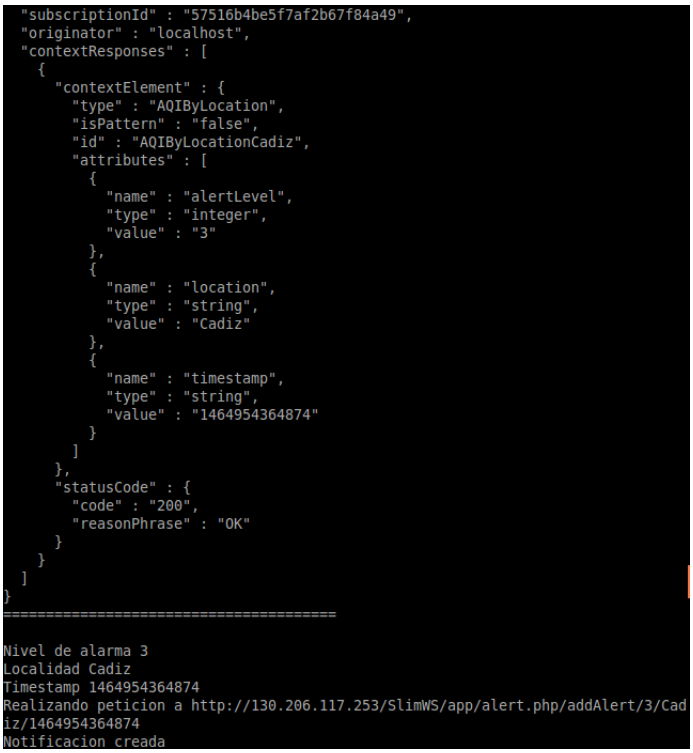

Figura 5.48.: Alerta de nivel 3 en Cádiz Terminal Consumidor

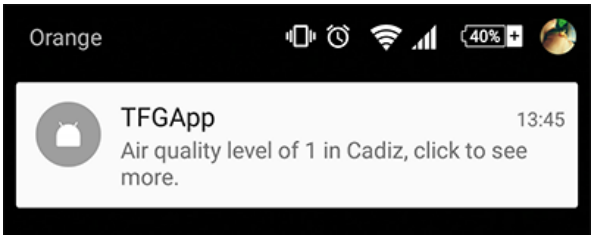

Figura 5.49.: Alerta de nivel 1 en Android

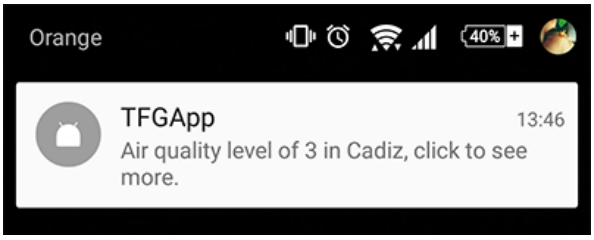

Figura 5.50.: Alerta de nivel 3 en Android

5.7. Problemas y dificultades encontradas

La mayoría de las dificultades a las que hemos tenido que hacer frente durante el desarrollo del proyecto han sido provocadas por el desconocimiento de esta novedosa plataforma que hemos empleado para el proyecto, FIWARE. Las primeras fases del proyecto que comprendían trabajos de investigación podemos identificarlas como las más duras, ya que al ser una herramienta que prácticamente acaba de nacer la documentación que podemos encontrar en línea es prácticamente escasa. Una vez conseguimos, en la medida de lo posible, dominar los diferentes aspectos de la plataforma como tal, los problemas que empezaron a aparecer estaban asociados a los componentes que empleábamos, Cepheus y Orion. Aunque ambos tienen una buena documentación oficial, hay ciertos aspectos que no están cubiertos o bien no se detallan tan bien como se debería, como la instalación conjunta, puesta en funcionamiento, etc. Estos aspectos los fuimos solventando gracias al ensayo y error.

Otro problema derivado de los componentes, en particular de Cepheus, fue en relación con los patrones de eventos complejos. Como ya hemos comentado anteriormente, la sintaxis varía un poco de EPL de Esper a la que debemos usar en el fichero de configuración de Cepheus. Además, un aspecto muy importante a notar y que originó el mayor problema, fue la necesidad de registrar un ID (similar a una clave primaria de SQL) para los eventos complejos que detectásemos en Cepheus.

Por otro lado, podemos destacar algunas pequeñas dificultades que hemos tenido a la hora de desarrollar el código Android para la aplicación, como el uso de AlarmManager para programar los disparadores en los diferentes rangos horarios. Aunque la documentación online de Android es mucha y, en general, muy buena, hay ciertos aspectos que Google cambia constantemente y uno de ellos, al parecer, es la clase AlarmManager. Son muchos los tutoriales que hemos tenido que consultar hasta, finalmente, lograr un correcto funcionamiento y conseguir implementar la funcionalidad que perseguíamos.

6. Conclusiones y trabajo futuro

En este capítulo trataremos las conclusiones a las que hemos llegado al término del proyecto y algunas recomendaciones de cara a una continuación futura de este proyecto.

6.1. Conclusiones

En este TFG hemos desarrollado un sistema en la nube capaz de detectar, en tiempo real, una serie de eventos complejos relativos a la calidad del aire; el sistema, posteriormente, notifica a una serie de usuarios finales la calidad del aire en cada instante mediante el uso de una aplicación Android. El TFG se ha desarrollado bajo la supervisión de los profesores Juan Boubeta-Puig y Guadalupe Ortiz, ambos pertenecientes al departamento de Ingeniería Informática de la Universidad de Cádiz y miembros del Grupo de Investigación UCASE de Ingeniería del Software (TIC-025).

Para la detección de patrones de eventos correspondientes a calidades del aire nocivas para la salud hemos empleado la plataforma FIWARE, una plataforma de despliegue de aplicaciones en la nube, dentro de la cual hemos empleado el componente Cepheus CEP para la detección de eventos complejos y Orion Context Broker para la gestión de contextos. Por otro lado, la aplicación final que empleará el usuario ha sido desarrollada en Android y es la encargada de monitorizar el contexto del usuario, es decir, la actividad que realiza y la localización, además de tener en cuenta sus propias preferencias a la hora de ser notificado y sus condiciones personales, como la edad y si padece o no problemas respiratorios.

Teniendo en cuenta los objetivos que nos marcamos al principio del proyecto, podemos concluir que:

- Hemos logrado detectar una serie de eventos complejos en tiempo real. Estos eventos complejos se identifican con los respectivos al nivel de calidad del aire. Estos eventos complejos son detectados en tiempo real y notificados al instante al usuario.

- Hemos conseguido realizar una aplicación funcional que lleva a cabo la monitorización del contexto del usuario en cada instante y, además, provee al usuario

de las herramientas necesarias para consultar información concerniente a su ubicación.

- Gracias a la monitorización del contexto del usuario, podemos afirmar que las notificaciones que éste recibe son acordes al contexto en el que se encuentra en dicho momento.
- Hemos logrado almacenar y llevar un registro de los diferentes eventos complejos detectados por el sistema.
- Hemos logrado ampliar conocimientos y dominar nuevos métodos de trabajo, herramientas y tecnologías tales como computación en la nube, detección de eventos complejos en tiempo real, programación Android, servicios web.

6.2. Trabajo futuro

El sistema que hemos conseguido crear presenta un funcionamiento estable y de buen rendimiento, aún así, existen una serie de mejoras que se podrían llevar a cabo con el objetivo de aumentar funcionalidades y mejorar otras existentes.

- **Implementación del módulo Cosmos de FIWARE.** Como ya mencionamos en capítulos anteriores, Cosmos es otro componente existente en el catálogo de FIWARE que realiza funciones de análisis de Big Data. Debido a que la cantidad de información que los sensores se encuentren suministrando a este sistema puede ser muy elevada, sería interesante estudiar la posibilidad de incorporar Cosmos al sistema con el objetivo de realizar tareas de Big Data, tales como informes de calidad del aire que nos ayuden en la toma de decisiones a la hora de llevar a cabo acciones que afecten a la calidad del aire de forma directa.
- **Implementación de un cuadro de mandos o *Dashboard*.** Otra utilidad muy interesante sería implementar algún tipo de *Dashboard* o *Scorecard* que permita una visualización de la información crítica que tenemos almacenada en Orion Context Broker de forma gráfica, intuitiva y atractiva. De esta forma, podríamos proveer a los usuarios de una aplicación web para consultar la calidad en un mapa o similar.
- **Mejoras en la monitorización.** A la hora de monitorizar al usuario disponemos de dos modos fundamentales, uno que es haciendo uso del GPS, en el cual una posible mejora sería lograr que el consumo de batería del dispositivo sea menor y aumentar el rendimiento y precisión de la geolocalización. La otra forma de ser monitorizado es mediante el uso de un horario predefinido por el

usuario, una mejora que podríamos implementar serían más periodos horarios durante el día y la posibilidad de escoger diferentes ubicaciones para diferentes periodos horarios.

A. Manual de instalación

En esta sección indicaremos los diferentes pasos para realizar la instalación del sistema de detección de eventos complejos y, por otro lado, la instalación de la aplicación Android.

A.1. Manual de instalación FIWARE

En este apartado detallaremos todo el proceso necesario para instalar los diferentes componentes que integran nuestra plataforma FIWARE. Para la instalación de este software, hemos empleado una máquina virtual con el sistema operativo CentOs 6.3 http://mirror.nsc.liu.se/centos-store/6.3/isos/x86_64/CentOS-6.3-x86_64-LiveDVD.iso que cuenta con una interfaz gráfica que será más cómoda a la hora de operar con diferentes terminales. Para realizar los siguientes pasos es altamente recomendable hacerlo como usuario root (ejecutar comando "su") y llevarlos a cabo en la carpeta raiz de root (si estamos como root: "cd ~/"; si no estamos como root: "cd /root/").

A.1.1. Instalación Orion Context Broker

Los pasos a seguir recogidos en este tutorial han sido obtenidos de la documentación oficial de Orion Context Broker [35].

1. Antes de comenzar con la instalación, es necesario definir el repositorio de FIWARE para la obtención de sus paquetes. Por lo tanto, accedemos a la carpeta de repositorios.

   ```
   cd /etc/yum.repos.d/
   ```

2. Creamos el fichero "fiware.repo" con las siguientes líneas.

   ```
   [fiware]
   name=Fiware Repository
   baseurl=http://repositories.lab.fiware.org/repo/rpm/$releasever
   gpgcheck=0
   enabled=1
   ```

3. Instalamos Orion haciendo uso del repositorio que acabamos de configurar.

```
yum install contextBroker
```

4. Instalamos herramientas para compilar y otras tareas.

```
sudo yum install make cmake gcc-c++ scons
```

5. Instalamos las librerías necesarias.

```
sudo yum install boost-devel libcurl-devel
   gnutls-devel libgcrypt-devel libuuid-devel
```

6. Instalamos los drivers necesarios para MongoDB desde la fuente.

```
wget https://github.com/mongodb/mongo-cxx-driver/archive/
         legacy-1.0.7.tar.gz
tar xfvz legacy-1.0.7.tar.gz
cd mongo-cxx-driver-legacy-1.0.7
scons
install --prefix=/usr/local
```

7. Instalamos *rapidjson* desde la fuente.

```
wget https://github.com/miloyip/rapidjson/archive/v1.0.2.tar.gz
tar xfvz v1.0.2.tar.gz
sudo mv rapidjson-1.0.2/include/rapidjson/ /usr/local/include
```

8. Instalamos *libmicrohttpd* desde la fuente.

```
wget http://ftp.gnu.org/gnu/libmicrohttpd/libmicrohttpd-0.9.48.tar.gz
tar xvf libmicrohttpd-0.9.48.tar.gz
cd libmicrohttpd-0.9.48
./configure --disable-messages --disable-postprocessor --disable-dauth
make
sudo make install
sudo ldconfig
```

9. Instalamos Google Test/Mock desde la fuente.

```
wget http://googlemock.googlecode.com/files/gmock-1.5.0.tar.bz2
tar xfvj gmock-1.5.0.tar.bz2
cd gmock-1.5.0
./configure make
sudo make install
sudo ldconfig
```

10. Instalar git para obtener el repositorio de Orion.

```
sudo yum install git
```

11. Descargar el repositorio de Orion. Se recuerda que es altamente recomendable hacerlo en la carpeta raiz de root, es decir, en /root/.

    ```
    git clone https://github.com/telefonicaid/fiware-orion
    ```

12. Construir la fuente de Orion.

    ```
    cd fiware-orion
    make
    ```

13. Opcional, pero altamente recomendable, ejecutar las unidades de prueba (dentro de la propia carpeta /fiware-orion/).

    ```
    sudo yum install mongodb-server
    sudo yum update pcre
    sudo /etc/init.d/mongod start
    sudo /etc/init.d/mongod status
    make unit_test
    ```

14. Instalamos el binario (dentro de la propia carpeta /fiware-orion/).

    ```
    sudo make install INSTALL_DIR=/usr
    ```

15. Finalmente nos aseguramos que Orion se ha instalado correctamente.

    ```
    contextBroker --version
    ```

A.1.2. Instalación Cepheus

Antes de instalar Cepheus CEP y Cepheus Broker será necesario tener instalado Java y Maven.

A.1.2.1. Instalar Javan y Maven

En primer lugar instalamos Java ya que es necesario para la instalación de Maven.

1. Descargamos la última versión de Java SE Development Kit 8 desde su página oficial.

    ```
    Para 64Bit
    ----------
    cd /opt/

    wget --no-cookies --no-check-certificate --header
    "Cookie: gpw_e24=http%3A%2F%2Fwww.oracle.com%2F;
    ```

```
oraclelicense=accept-securebackup-cookie"
"http://download.oracle.com/otn-pub/java/jdk/
        8u91-b14/jdk-8u91-linux-x64.tar.gz"

tar xzf jdk-8u91-linux-x64.tar.gz


Para 32Bit
----------
cd /opt/

wget --no-cookies --no-check-certificate --header
"Cookie: gpw_e24=http%3A%2F%2Fwww.oracle.com%2F;
oraclelicense=accept-securebackup-cookie"
"http://download.oracle.com/otn-pub/java/jdk/
        8u91-b14/jdk-8u91-linux-i586.tar.gz"

tar xzf jdk-8u91-linux-i586.tar.gz
```

2. Ahora vamos a instalar Java usando *alternatives*.

```
cd /opt/jdk1.8.0_91/
alternatives --install /usr/bin/java java /opt/jdk1.8.0_91/bin/java 2
alternatives --config java

# En este punto seleccionamos el que presente
# el comando/ruta: /opt/jdk1.8.0_91/bin/java
```

3. En este punto, tendremos Java 8 instalado. Aplicamos el *path* de javac and jar usando estos comandos.

```
alternatives --install /usr/bin/jar jar /opt/jdk1.8.0_91/bin/jar 2
alternatives --install /usr/bin/javac javac /opt/jdk1.8.0_91/bin/javac 2
alternatives --set jar /opt/jdk1.8.0_91/bin/jar
alternatives --set javac /opt/jdk1.8.0_91/bin/javac
```

4. Comprobamos la versión instalada.

```
java -version
```

5. Finalmente aplicamos las variables de entorno con los siguientes comandos.

```
export JAVA_HOME=/opt/jdk1.8.0_91
export JRE_HOME=/opt/jdk1.8.0_91/jre
export PATH=$PATH:/opt/jdk1.8.0_91/bin:/opt/jdk1.8.0_91/jre/bin
```

Ahora vamos a instalar Maven.

1. En primer lugar nos descargamos la distribución de Maven que deseemos instalar de http://maven.apache.org/download.cgi. En este caso emplearemos la versión Apache Maven 3.3.9.

   ```
   wget http://ftp.cixug.es/apache/maven/maven-3/3.3.9/binaries/
           apache-maven-3.3.9-bin.tar.gz
   ```

2. Extraemos el contenido y lo movemos al directorio /opt/.

   ```
   tar xfv apache-maven-3.3.9-bin.tar.gz
   mv apache-maven-3.3.9/ /opt/maven
   ```

3. Creamos un enlace simbólico a la carpeta de maven.

   ```
   ln -s /opt/maven/bin/mvn /usr/bin/mvn
   ```

4. Para fijar la variable de entorno de Maven, vamos a crear un fichero denominado maven.sh en el directorio /etc/profile.d/.

   ```
   vi /etc/profile.d/maven.sh
   ```

5. A dicho fichero, le añadimos las siguientes líneas.

   ```
   #!/bin/bash
   MAVEN_HOME=/opt/maven
   PATH=$MAVEN_HOME/bin:$PATH
   export PATH MAVEN_HOME
   export CLASSPATH=.
   ```

6. Guardamos el fichero y le aplicamos el permiso de ejecución.

   ```
   chmod +x /etc/profile.d/maven.sh
   ```

7. Fijamos la variable de entorno con el siguiente comando.

   ```
   source /etc/profile.d/maven.sh
   ```

8. Comprobamos la versión de Maven instalada.

   ```
   mvn -version
   ```

9. Reiniciamos el ordenador o la máquina virtual para que todos estos cambios surtan efecto.

A.1.2.2. Instalar Cepheus CEP y Cepheus Broker

A continuación mostramos los diferentes pasos a seguir para instalar el componente Cepheus al completo. Es altamente recomendable ejecutar estos pasos como usuario root desde la carpeta /root/. En cualquier caso, es recomendable instalar Cepheus en la misma carpeta que hemos empleado para instalar Orion. Los pasos para la instalación de Cepheus han sido obtenidos de la documentación oficial [34].

1. Descargamos el repositorio Git del proyecto Cepheus.

   ```
   git clone https://github.com/Orange-OpenSource/fiware-cepheus.git
   ```

2. Accemos a la carpeta de Cepheus CEP.

   ```
   cd fiware-cepheus/cepheus-cep
   ```

3. A continuación construimos la fuente.

   ```
   mvn clean package
   ```

4. Comprobamos que Cepheus CEP se ha instalado correctamente ejecutándolo.

   ```
   mvn spring-boot:run
   ```

5. Para detenerlo, pulsamos Control + C.

Una vez instalado Cepheus CEP, pasamos a intalar Cepheus Broker.

1. Ya que el repositorio Git lo hemos descargado previamente para instalar Cepheus CEP, accedemos a la carpeta de Cepheus Broker.

   ```
   cd fiware-cepheus/cepheus-broker
   ```

2. A continuación construimos la fuente.

   ```
   mvn clean package
   ```

3. Descargamos version .jar del broker.

   ```
   wget -O cepheus-broker.jar "https://oss.sonatype.org/service/local/
           artifact/maven/redirect?r=snapshots&g=com.orange.cepheus&a=
           cepheus-broker&v=LATEST"
   ```

4. Comprobamos que Cepheus Broker se ha instalado correctamente ejecutándolo.

   ```
   java -jar cepheus-broker.jar --remote.url="http://localhost:1026"
   ```

5. Para detenerlo, pulsamos Control + C.

A.1.3. Instalación de Python, Pip y Flask

Finalmente, para la ejecución del consumidor de eventos que está escrito en el lenguaje de programación Python es necesario instalar una serie de librerias auxiliares.

1. En primer lugar nos aseguramos de tener instalado GCC en el sistema.

   ```
   yum install gcc
   ```

2. A continuación instalamos Python. Para este manual empleamos la versión 2.7.

   ```
   cd /usr/src
   wget https://www.python.org/ftp/python/2.7.10/Python-2.7.10.tgz
   ```

3. Extraemos el fichero y lo compilamos.

   ```
   tar xzf Python-2.7.10.tgz
   cd Python-2.7.10
   ./configure
   make altinstall
   ```

4. Comprobar que Python se ha instalado correctamente.

   ```
   python2.7 -V
   ```

Una vez instalado Python, pasamos a instalar la libreria auxiliar pip.

1. Ejecutamos el comando correspondiente al sistema que empleemos.

   ```
   Para 64Bit
   ----------
   rpm -ivh http://dl.fedoraproject.org/pub/epel/6/x86_64/
                   epel-release-6-8.noarch.rpm

   Para 32Bit
   ----------
   rpm -ivh http://dl.fedoraproject.org/pub/epel/6/i386/
                   epel-release-6-8.noarch.rpm
   ```

2. Instalamos pip con el comando yum.

   ```
   yum install -y python-pip
   ```

Finalmente, instalamos la librería Flask.

1. Antes de instalar Flask es necesario asegurarnos de que tenemos instalado virtualenv.

```
sudo pip install virtualenv
```

2. Ejecutamos el siguiente comando para instalar Flask.

   ```
   pip install Flask
   ```

Llegamos a este punto, todas las librerias y los paquetes necesarios para ejecutar nuestro proyecto estarán instalados y presentes en el sistema.

A.2. Manual de instalación Android

A continuación, detallaremos los diferentes pasos necesarios para instalar la aplicación Android en un terminal. El único requisito indispensable es que la versión Android del dispositivo móvil sea igual o superior a Android 4.1 y tener en cuenta las siguientes recomendaciones:

- Tenemos la aplicación con extensión .apk que vamos a instalar.
- Tenemos el terminal conectado al ordenador vía USB.
- Tenemos activada la opción para instalar aplicaciones de orígenes desconocidos en el terminal que deseemos instalarlo. Para activarlo, basta con acceder a "Ajustes" en el terminal, a continuación acceder a "Seguridad" y activar la pestaña de "Orígenes desconocidos - Permitir la instalación de aplicaciones de origen desconocido".
- Es altamente recomendable tener activadas las opciones de desarrollador y la depuración USB. Activar las opciones del desarrollador depende del terminal, pero por norma general basta con hacer entre 4 y 7 clicks sobre la versión de Android del dispositivo que podemos encontrar en Ajustes. Por otro lado, la depuración USB la activaremos dentro de las opciones de desarrollador.

1. En primer lugar, accedemos a https://apkpure.com/apk-install.html para descargar el software de Windows "Pure APK Install" que nos permitirá instalar aplicaciones Android en nuestro terminal de forma muy sencilla.

   a) Pulsamos sobre el botón *"Download Now"*.

2. A continuación instalamos el software descargado (Pure_APK_Install_setup.exe) aceptando los diferentes pasos.

3. Una vez instalado, lo ejecutamos.

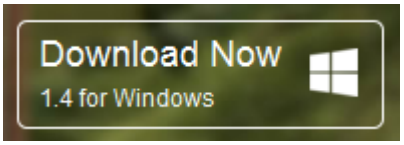

Figura A.1.: Botón para descargar “Pure APK Install”.

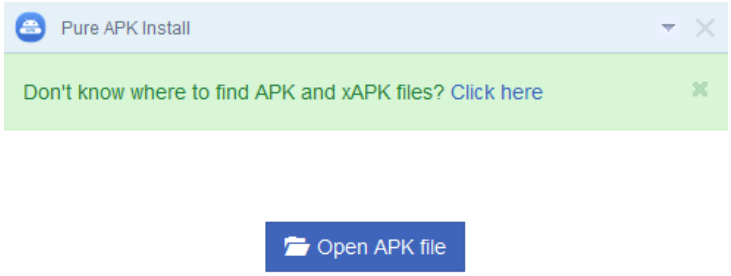

Figura A.2.: Programa “Pure APK Install” iniciado.

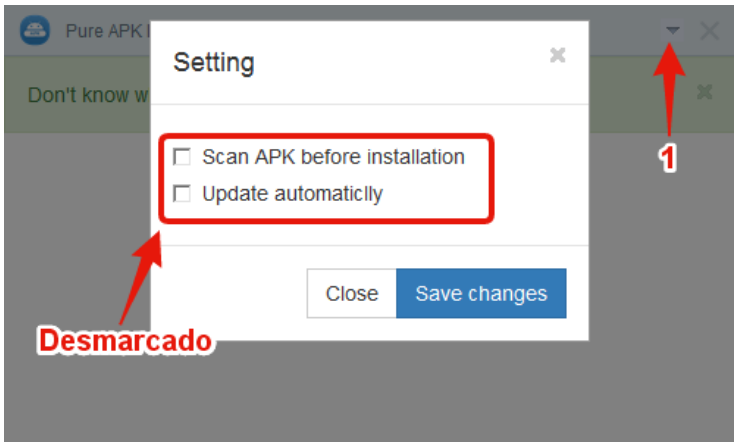

Figura A.3.: Ajustes en “Pure APK Install”.

4. Pulsamos sobre la flecha que está señalada como "1", figura A.3, y a continuación desmarcamos los campos *Scan APK before installation* y *Update automatically*. Finalmente pulsamos sobre *Save changes* para guardar estos cambios.

5. Ahora pulsamos sobre el botón azul *Open APK file* y buscamos el fichero .apk de la aplicación Android que deseamos instalar. En este caso se denomina app-Air-Quality.apk, figura A.4.

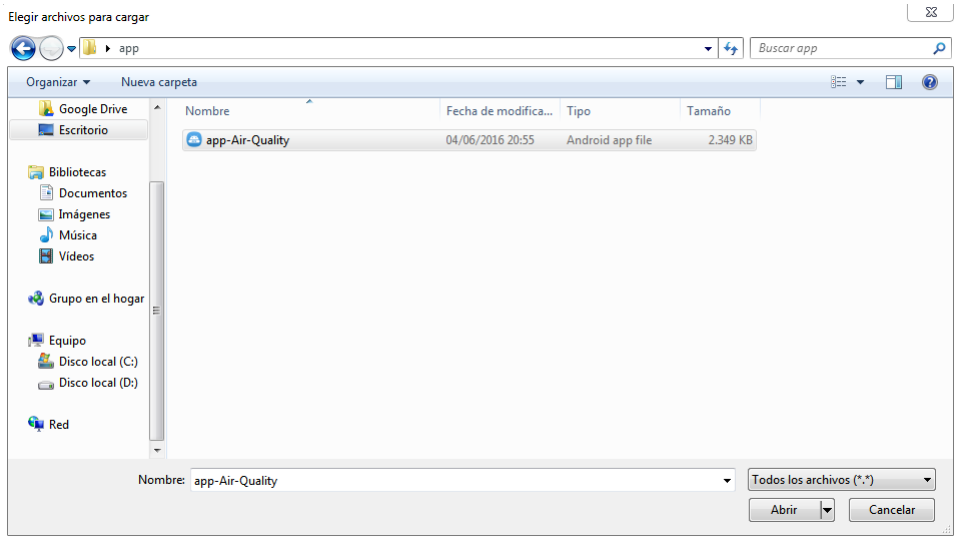

Figura A.4.: Seleccionamos el apk a instalar.

6. Al seleccionar el apk se mostrará la información de la aplicación que vamos a instalar: Nombre, peso, versión de la app, versión de Android mínima y dónde deseas instalarlo. En el campo *You want to install this APK to* seleccionamos *Internal android memory* para instalar la aplicación en la memoria interna del terminal, figura A.5.

7. Pulsamos sobre *Install* y comenzará la instalación de la aplicación en el terminal. **Asegúrese de que el terminal está conectado mediante USB al ordenador.** En algún momento del proceso de instalación puede aparecer un aviso en el terminal para confirmar la instalación de la aplicación, lo confirmaremos si fuese necesario.

8. Si todo ha ido bien, el programa nos mostrará un mensaje de éxito, figura A.7, y tendremos la app instalada en el terminal, figura A.8.

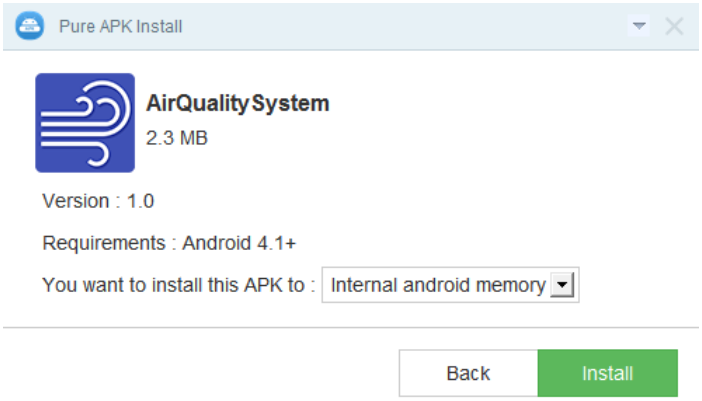

Figura A.5.: Información del apk a instalar.

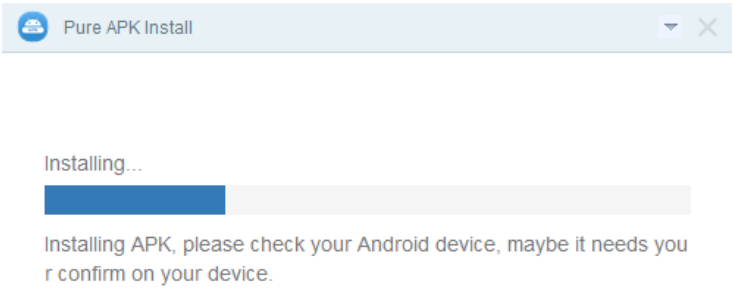

Figura A.6.: Instalación del apk.

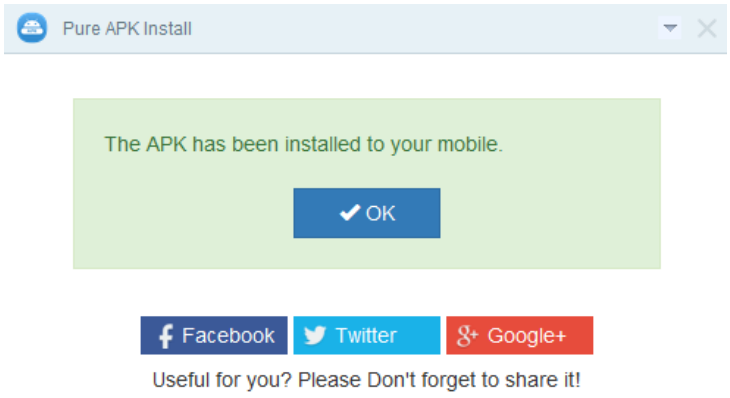

Figura A.7.: Instalación del apk correcta.

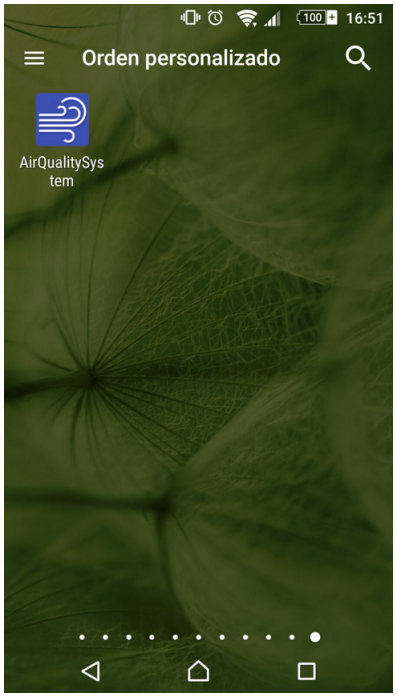

Figura A.8.: Aplicación instalada en el terminal.

B. Manual de usuario

En esta sección visualizaremos, por un lado, el manual de usuario de la plataforma FIWARE, necesario para desplegar cualquier proyecto y, por otro lado, el manual de usuario de la aplicación Android que abarcará las diferentes funcionalidades que posee la aplicación.

B.1. Manual de usuario FIWARE

A continuación vamos a ilustrar el proceso para que, una vez instalado el sistema, podamos ejecutarlo y hacer uso de sus funcionalidades. Antes de comenzar es recomendable usar algún software que nos permita emplear varias terminales al mismo tiempo de forma compacta. Para el desarrollo de este manual hemos empleado Terminator.

1. Ejecutar en una primera terminal Orion Context Broker.

   ```
   [root@system ~]# /etc/init.d/contextBroker start
   ```

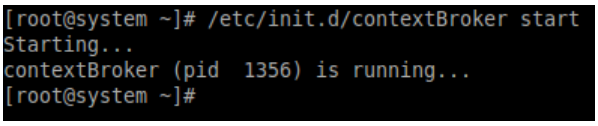

Figura B.1.: Ejecución Orion Context Broker

2. En la misma terminal que hemos empleado, desplegar el servidor LAMPP

   ```
   [root@system ~]# /opt/lampp/lampp start
   ```

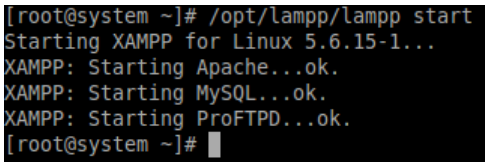

Figura B.2.: Despliegue servidor LAMPP

3. Ejecutar Cepheus Broker en la misma terminal que hemos empleado para iniciar Orion. En este caso especificamos el puerto en el cual se encuentra Orion Context Broker (1026).

```
[root@system ~]# cd fiware-cepheus/cepheus-broker/
[root@system cepheus-broker]# java -jar cepheus-broker.jar
        --remote.url="http://localhost:1026"
```

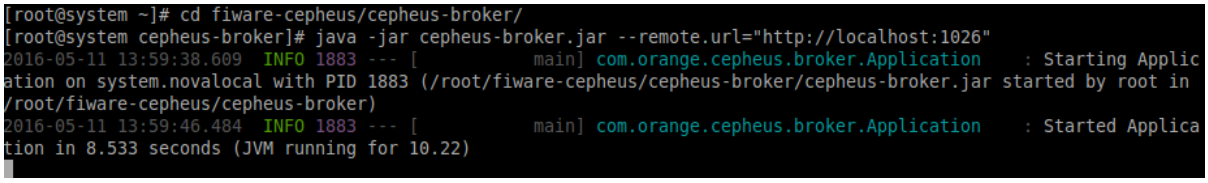

Figura B.3.: Ejecución Cepheus Broker

4. En una segunda terminal ejecutaremos Cepheus CEP haciendo uso de maven.

```
[root@system ~]# cd fiware-cepheus/cepheus-cep/
[root@system cepheus-cep]# mvn spring-boot:run
```

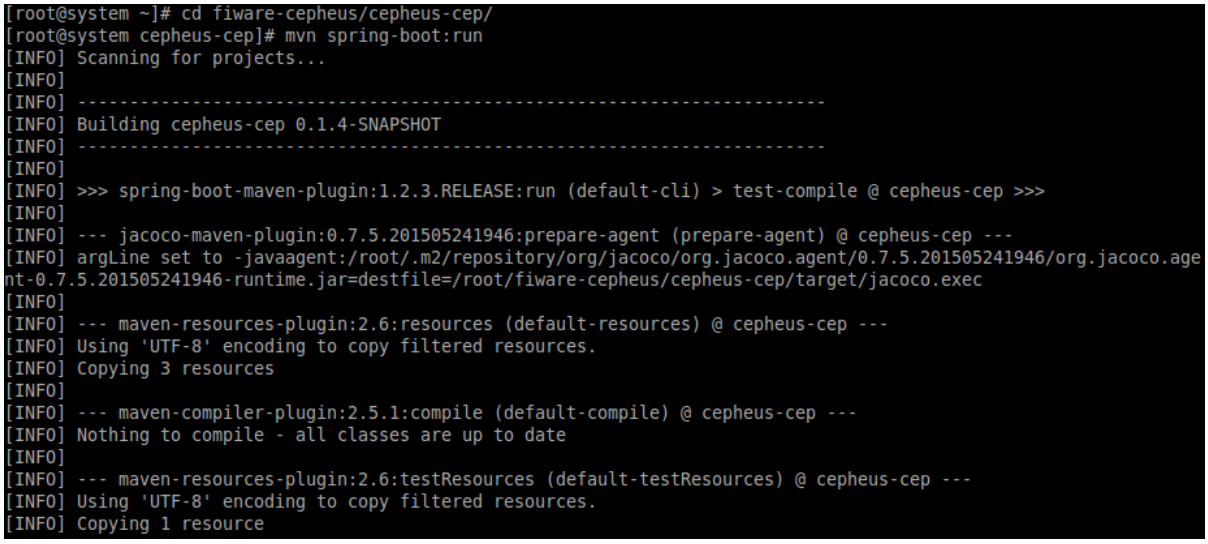

Figura B.4.: Ejecución Cepheus CEP

Si hemos ejecutado el sistema, más especificamente Cepheus CEP, anteriormente, éste cargará la última configuración empleada. En caso contrario, será necesario aplicar la configuración deseada al motor CEP, para ello ejecutamos el siguiente comando en la misma ubicación donde se encuentre el fichero de configuración, en este caso config.json

```
cat config.json | curl -H 'Accept: application/json' -H \
'Content-Type: application/json' -d @-
        http://localhost:8080/v1/admin/config
```

5. A continuación ejecutaremos el consumidor de eventos Python, en una nueva terminal, con especial atención al puerto dónde decidamos ejecutarlos, en este caso el 1028.

```
[root@system ~]# cd airsensor/system/
[root@system system]# ./accumulator-server.py 1028 /accumulate on
```

```
[root@system system]# ./accumulator-server.py 1028 /accumulate ::1 on
verbose mode is on
PID file /tmp/accumulator.1028.pid already exists, killing the process 4108
Process 4108 killed
```

Figura B.5.: Ejecución consumidor AQI Levels

6. Ahora será necesario ejecutar las condiciones de suscripción para cada uno de los diferentes consumidores que hayamos decidido ejecutar. En nuestro caso hemos ejecutado el de alertas de calidad del aire en el puerto 1028. Por lo tanto, en una nueva terminal ejecutamos el siguiente comando.

```
### PUERTO 1028 PARA ALERTAS DE CALIDAD DEL AIRE ###
(curl localhost:1026/v1/subscribeContext -s -S --header 'Content-Type:
application/json' --header 'Accept: application/json' -d @- |
python -mjson.tool) <<EOF
{
    "entities": [
        {
            "type": "AQIByLocation",
            "isPattern": "true",
            "id": "AQIByLocation.*"
        }
    ],
    "attributes": [
        "alertLevel",
        "location",
        "timestamp"
    ],
    "reference": "http://localhost:1028/accumulate",
    "duration": "PT24H",
    "notifyConditions": [
```

```
            {
                "type": "ONCHANGE",
                "condValues": [
                    "alertLevel"
                ]
            }
        ]
    }
    EOF
```

7. Llegados a este punto, nuestro sistema estará preparado para comenzar su funcionamiento. Notar que las condiciones de suscripción que hemos empleado en el paso anterior tienen una validez de 24 horas. En caso de que queramos emplear otro tiempo de validez de esta suscripción, bastará con modificar el campo *duration* indicando el tiempo deseado según las normas descritas en el estándar ISO 8601 [19].

B.2. Manual de usuario Android

A continuación mostraremos de forma detallada las diferentes funcionalidades de la aplicación Android.

B.2.1. Menú de la aplicación

El menú de la aplicación está disponible al pulsar sobre los tres puntos blancos que están en la esquina superior derecha.

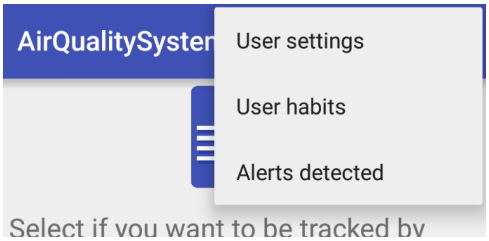

Figura B.6.: Menú de la aplicación

B.2.2. Pantalla de inicio

Al entrar en la aplicación veremos dos botones que nos redireccionarán a los dos modos de seguimiento que presenta la aplicación, véase la Figura B.7

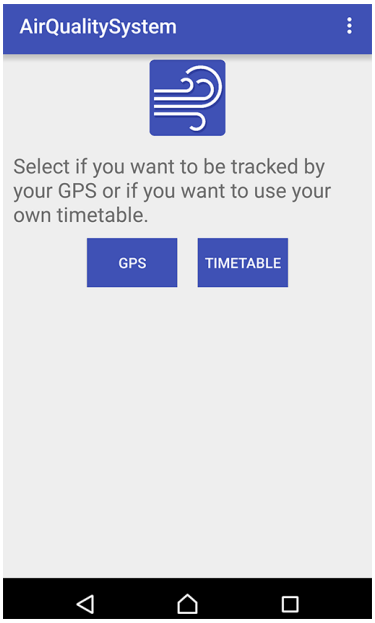

Figura B.7.: Pantalla inicial aplicación

B.2.3. Pantalla de ajustes

Para acceder a ajustes, pulsamos para abrir el menú y seleccionamos la opción *"User settings"*. En este apartado tenemos la opción a configurar el tipo de notificaciones que deseamos recibir y si padecemos o no algún tipo de enfermedad respiratoria marcando o desmarcando el checkbox correspondiente. Por otro lado, en el menú desplegable tenemos la opción de elegir nuestro sango de edad. Para guardar los cambios es necesario hacer click en el botón *"Finish"*, véase Figura B.8.

B.2.4. Pantalla de hábitos

Para acceder a los hábitos u horario establecido por el usuario, pulsamos para abrir el menú y seleccionamos la opción *"User habits"*. En este apartado tenemos la opción de definir el horario habitual que seguimos para cada día de la semana, véase Figura B.9.

Para editar o definir el horario de un día, será necesario pulsar sobre el botón *"Edit"*, véase Figura B.10, en el día que deseemos definir o editar. Para cambiar de día, hacemos uso de las flechas que están a izquierda y derecha del nombre del día.

Una vez dentro del día que deseamos editar, seleccionamos el tipo de actividad que desempeñamos ese día de la semana en ese periodo de tiempo, véase Figura B.11, y, al final, seleccionamos la provincia dónde nos encontremos durante ese día y guardamos los cambios pulsando sobre el botón *"Finish"*, véase Figura B.12.

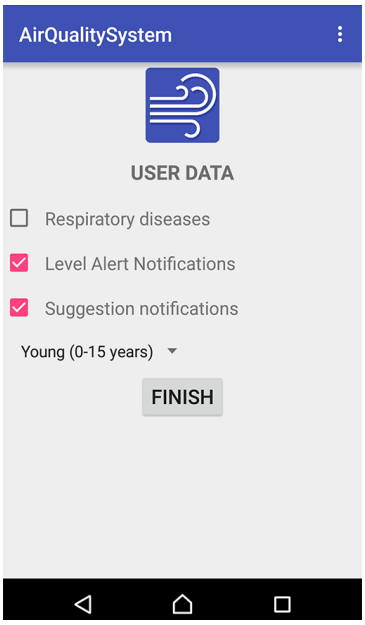

Figura B.8.: Pantalla de ajustes aplicación

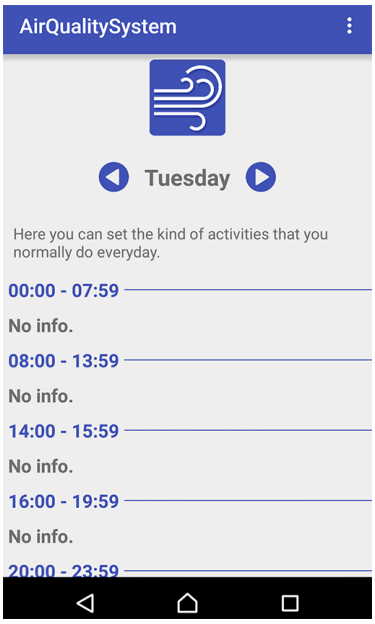

Figura B.9.: Pantalla de horario aplicación

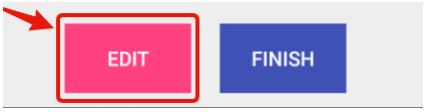

Figura B.10.: Botón editar horario

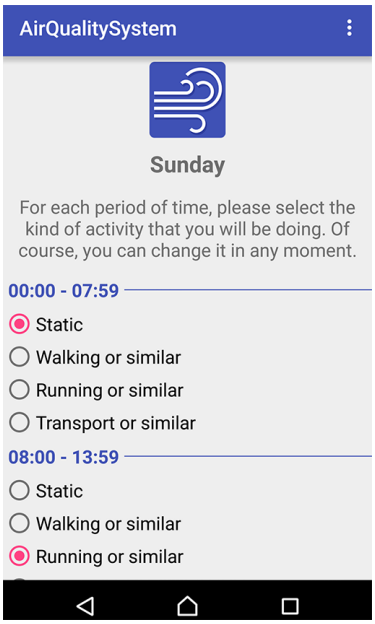

Figura B.11.: Escoger actividades en horario aplicación

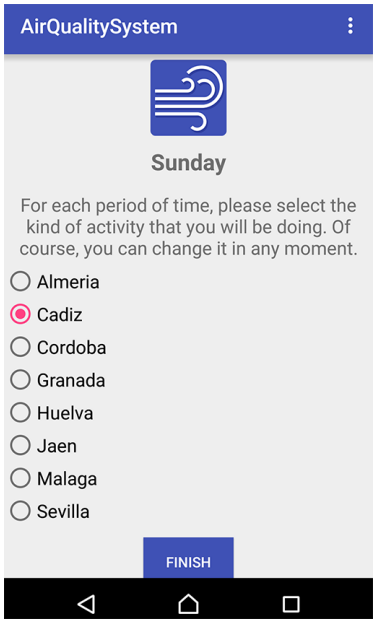

Figura B.12.: Escoger provincia en horario aplicación

B.2.5. Pantalla de alertas

Para acceder a las alertas detectadas en el área donde te encuentras o has definido en tu horario, pulsamos para abrir el menú y seleccionamos la opción *"Alerts detected"*. Automáticamente se tomará la ubicación en función del GPS, si se encuentra activado o, en caso contrario, la obtendrá del horario que hemos definido en el apartado para ello. Si no está tampoco definido, no se mostrarán las alertas.

Una vez con la ubicación definida, se mostrarán las últimas 5 alertas detectadas para esa ubicación. Destacar que la primera siempre será el nivel actual en dicha localidad, véase Figura B.13.

B.2.6. Pantalla de monitorización por GPS

Para acceder al apartado de monitorización por horario, abrimos la aplicación y pulsamos sobre el botón *"GPS"* de la pantalla inicial. El único requisito es tener el sensor GPS habilitado, en tal caso la monitorización comenzará al instante, véase Figura B.14.

En caso de no tener el GPS activado, la aplicación mostrará un mensaje de error, véase Figura B.15.

Para finalizar la monitorización mediante GPS cerramos la aplicación y la monitorización automáticamente se abortará.

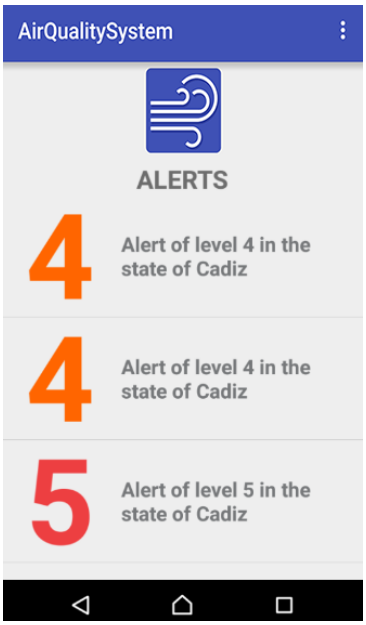

Figura B.13.: Mostrar alertas detectadas aplicación

Figura B.14.: Monitorización por GPS aplicación

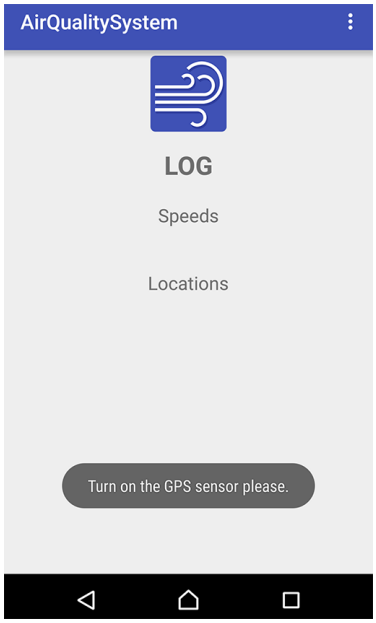

Figura B.15.: Mensaje error GPS no activado

B.2.7. Pantalla de monitorización por horario

Para acceder al apartado de monitorización por horario, abrimos la aplicación y pulsamos sobre el botón *"Timetable"* de la pantalla inicial. Si tenemos el horario definido con anterioridad para el día en el que deseamos realizar la monitorización, esta comenzará automáticamente, véase Figura B.16.

En caso de que no hayamos definido el horario para dicho día, la aplicación nos mostrará un aviso, véase Figura B.17.

Para finalizar la monitorización, pulsamos sobre el botón *"Stop"* y la aplicación dejará de monitorizar nuestros cambios de contextos, véase Figura B.18. Para volver al menú principal, pulsamos sobre el botón *"Ok"*, véase Figura B.18.

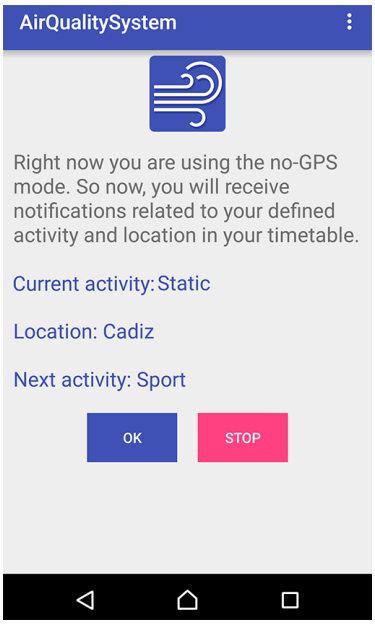

Figura B.16.: Monitorización por horario aplicación

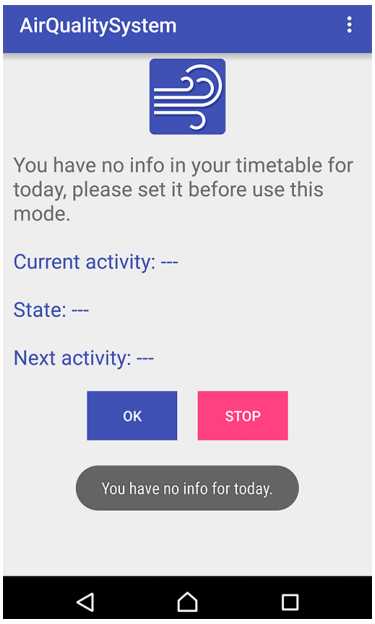

Figura B.17.: Mensaje de error por horario no definido

Figura B.18.: Botones para volver al menú principal y cancelar monitorización

C. Manual del desarrollador

En este anexo vamos a proporcionar las instrucciones necesarias que permitirán que cualquier desarrollador interesado en el proyecto pueda seguir mejorándolo y ampliando sus funcionalidades.

C.1. Modificar fichero de configuración

Modificar el fichero de configuración puede ser necesario para editar o definir nuevos eventos simples, eventos complejos y patrones de eventos. Por lo tanto dividiremos esta sección en tres partes. Para más información con respecto al fichero de configuración, podemos consultar la documentación oficial: http://fiware-cepheus.readthedocs.io/en/latest/cep/configuration/index.html

C.1.1. Añadir o modificar un evento simple

Añadir un nuevo evento simple es muy sencillo, solo será necesario:

1. Abrir el fichero de configuración, generalmente config.json, con algún editor de texto.
2. Definir el esquema del evento simple, para ello podemos hacer uso de este esquema general y adaptarlo a nuestra situación. Los campos que aparecen en el esquema se corresponden con:
   a) ***id***. El id del evento simple, puede seguir un patrón (*Room1, Room2, Room3...*)
   b) ***type***. El tipo del evento simple.
   c) ***isPattern***. Verdadero si el id sigue un patrón, falso en caso contrario.
   d) ***atributtes***. Los atributos de los que se compone el evento simple, estos pueden tener metadatos con su propio tipo particular.
3. Una vez tengamos nuestro esquema definido, bastará con agregarlo dentro del vector de objetos JSON en el campo ***in*** de nuestro fichero config.json.

Para modificar un evento simple, bastará con realizar las modificaciones pertinentes sobre el esquema ya codificado en el fichero de configuración.

C.1.2. Añadir o modificar un evento complejo

Añadir o modificar un evento complejo es muy similar a los eventos simples, solo cambian algunos campos con respecto a los simples. Por lo tanto, para añadir un nuevo evento complejo:

1. Abrir el fichero de configuración, generalmente config.json, con algún editor de texto.

2. Definir el esquema del evento complejo, para ello podemos hacer uso de este esquema general y adaptarlo a nuestra situación.

```
{
      "id":"FloorX",
      "type":"Floor",
      "attributes":[
        {
          "name":"temperature", "type":"double",
          "metadata": [
            { "name":"unit", "type":"string" }
          ]
        }
      ]
      "brokers":[
        {
          "url":"http://orion.fiware.org:3000",
          "serviceName": "tenant",
          "servicePath": "test/example",
          "authToken": "OAUTH_TOKEN"
        }
      ]
}
```

   a) ***id***. El id del evento complejo por defecto por si el patrón no define uno específico.

   b) ***type***. El tipo del evento complejo.

   c) ***atributtes***. Los atributos de los que se compone el evento complejos, estos pueden tener metadatos con su propio tipo particular.

   d) ***brokers***. Los brokers que serán notificados del evento complejo. Estos a su vez tienen una serie de parámetros, aunque el único obligatorio es la ***url.***

3. Una vez tengamos nuestro esquema definido, bastará con agregarlo dentro del vector de objetos JSON en el campo ***out*** de nuestro fichero config.json.

Para modificar un evento complejo, bastará con realizar las modificaciones pertinentes sobre el esquema ya codificado en el fichero de configuración.

C.1.3. Añadir o modificar un patrón de eventos complejos

Los patrones de eventos complejos se definen usando el lenguaje EPL de Esper. Para añadir un nuevo seguiremos los siguientes pasos:

1. Abrir el fichero de configuración, generalmente config.json, con algún editor de texto.

2. Definir el patrón de eventos complejos en cuestión, por ejemplo:

```
INSERT INTO Floor
SELECT floor as id, avg(temperature) as temperature
FROM Room.win:time(10 min)
GROUP BY floor
OUTPUT LAST EVERY 10 sec
```

3. Una vez definido, bastará con agregarlo entre comillas al campo ***statements*** de nuestro fichero config.json.

Para modificar uno existente, basta con realizar los cambios pertinentes directamente en el fichero de configuración. Una vez realizados dichos cambios, será necesario notificar al sistema de la nueva configuración mediante el comando para enviar la nueva configuración.

```
cat config.json | curl -H 'Accept: application/json' -H \
'Content-Type: application/json' -d @-
        http://localhost:8080/v1/admin/config
```

C.2. Consumidor de eventos

En el momento que agreguemos nuevos eventos complejos y sus correspondientes patrones, si deseamos que estos sean consumidos y posteriormente notificados será necesario crear un nuevo consumidor específico para este tipo de evento complejo que estamos tratando. Tal y como hemos visto en la sección de implementación, el consumidor que hemos empleado es el original, desarrollado por Fermín Galán Márquez, con una serie de modificaciones para que desempeñe el trabajo que deseamos realizar. El código del consumidor es muy extenso, se encuentra en el Anexo D.3,

pero la parte que realmente debemos modificar/alterar para que se traten eventos complejos se encuentra en una determinada sección del código.

Como ya hemos visto, cuando recibimos un evento complejo en el consumidor, lo que recibimos es un JSON con la información de dicho evento complejo que hemos solicitado al realizar la suscripción a Orion Context Broker. Por lo tanto, para realizar determinadas acciones solo tendremos que ser capaces de manejar ese JSON para obtener los datos. Esto, junto a unos conocimientos de Python sobre la tarea que deseemos desempeñar (invocar a un Servicio Web con los datos, almacenarlos en una base de datos, etc.) será todo lo necesario. El fragmento de código que decodifica el JSON e invoca al Servicio Web es el siguiente.

```
if verbose:
        #Pintamos el JSON recibido (aún una cadena de carácteres)
        print s
        #Eliminamos los saltos de línea
        valueJSON = valueJSON.replace('\n','')
        #Eliminamos los espacios en blanco
        valueJSON = valueJSON.replace(' ','')
        import json
        #Codificamos la cadena en formato JSON
        j = json.loads(valueJSON)
        #Mostramos los atributos y, a continuación, los almacenamos
        print "Nivel de alarma %s" %
            j['contextResponses'][0]['contextElement']['attributes'][0]['value']
        print "Localidad %s" %
            j['contextResponses'][0]['contextElement']['attributes'][1]['value']
        print "Timestamp %s" %
            j['contextResponses'][0]['contextElement']['attributes'][2]['value']
        nivel =
            j['contextResponses'][0]['contextElement']['attributes'][0]['value']
        localidad =
            j['contextResponses'][0]['contextElement']['attributes'][1]['value']
        idtime =
            j['contextResponses'][0]['contextElement']['attributes'][2]['value']
        url =
            "http://130.206.117.253/SlimWS/app/alert.php/addAlert/%i/%s/%s"
            % (int(nivel),localidad,idtime)
        print "Realizando peticion a %s" % url
        import urllib2
        #Petición GET al Servicio Web para que cree la notificación
        urllib2.urlopen(url).read()
```

```
print "Notificacion creada"
```

En este caso, como hemos podido apreciar en los comentarios del código lo que hacemos es que para cada evento complejo detectado, obtenemos los atributos que hemos indicado en la condición de suscripción y mediante una librería de Python, invocamos al Servicio Web que se encarga de crear la notificación para los usuarios y registrar el evento complejo en la base de datos. Para implementar cualquier otra acción solo será necesario saber realizarla en Python, como ya hemos comentado.

C.3. Servidor LAMPP

El sistema hace uso de un Servidor Apache LAMPP para desplegar la base de datos MySQL dónde registraremos los diferentes eventos complejos detectados y por otro lado que nos servirá para implementar los Servicios Web en el lenguaje PHP.

C.3.1. Base de datos MySQL

Las bases de datos del servidor LAMPP utilizan phpMyAdmin [51], que es una herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas web. Por lo tanto, al contar con una interfaz gráfica, es áltamente recomendable hacer uso de ella para realizar los cambios oportunos. De tal forma, para acceder a dicha interfaz gráfica:

1. Iniciaremos el servidor LAMPP con el siguiente comando.

   ```
   [root@system ~]# /opt/lampp/lampp start
   ```

2. A continuación, desde cualquier navegador accederemos a la dirección compuesta por la IP pública del servidor dónde tengamos hospedado nuestro sistema y la extesión phpMyAdmin. Siendo, por ejemplo: http://188.177.12.65/phpMyAdmin

3. A partir de este momento estaremos en el panel de control de la administración de la base de datos, dónde podremos realizar todas las acciones que atañen a la base de datos que usamos para registrar los eventos complejos detectados.

C.3.2. Servicios Web

Los Servicios Web que hemos empleado están implementados en PHP, haciendo uso de Slim Framework [63] que es un *framework* para desarrollar Servicios Web. Los diferentes Servicios Web, por normal general, se encuentran ubicados en:

```
/opt/lampp/htdocs/SlimWS/app/
```

La estructura que presentan es la siguiente:

```
<?php
require 'Slim/Slim.php';
\Slim\Slim::registerAutoloader();
$app = new \Slim\Slim();
$app->get(
      '/nombreWS/:par1/:par2/:par3', //Ruta con parámetros
      function ($par1, $par2, $par3){ //Parámetros en variables por orden
                        /* CUERPO WS */
          });
$app->run();
?>
```

Para crear uno nuevo bastaría con emplear el esquema superior y escribir el código necesario para desempeñar la funcionalidad deseada. Para modificar uno existente, basta con acceder al fichero PHP del Servicio Web que deseemos modificar y realizar los cambios pertinentes.

D. Códigos fuente

D.1. Servicios web

D.1.1. Alert

```
<?php
require 'Slim/Slim.php';
SlimSlim::registerAutoloader();
$app = new SlimSlim();
$app->get('/addAlert/:level/:location/:IDtimestamp',
function ($level, $location, $IDtimestamp)
{
  include_once "conection.php";
  $location = ucfirst(strtolower($location));
  $sql = "SELECT * FROM alert WHERE `IDtimestamp` = '" .
     $IDtimestamp . "' AND `location` = '" . $location . "'";
  $result = mysql_query($sql);
  $cant = mysql_num_rows($result);
  if ($cant == 0)
  {
        $sql = "INSERT INTO `alert`(`level`, `location`,
               `timestamp`, `IDtimestamp`) VALUES (" . $level . ",
               '" . $location . "','" . date("Y-m-d H:i:s") . "',
               '" . $IDtimestamp . "')";
        $result = mysql_query($sql);
        if (!$result)
        {
          echo "Connection ERROR = " . mysql_error();
          exit();
        }
        else
        {
        echo "Alert created correctly.<br/>";
        }
```

```
// Push The notification with parameters
require_once ('PushBots.class.php');
$pbS = new PushBots(); // Static
$pbW = new PushBots(); // Walking
$pbR = new PushBots(); // Running
$pbC = new PushBots(); // Car
// Application ID
$appID = '56a3b9c1177959ca6c8b4569';
// Application Secret
$appSecret = '191b456225dc6b0528ef98658a9acf0d';
$pbS->App($appID, $appSecret);
$pbW->App($appID, $appSecret);
$pbR->App($appID, $appSecret);
$pbC->App($appID, $appSecret);
// Setting the notification
switch ($level)
{
case 0:
 // Notification Settings
 // In this case, the notification is the same
 $tag = array(
       $location . "Static",
       $location . "Walking",
       $location . "Sport",
       $location . "Transport"
 );
 // Big Text
 $customfields = array(
       "BigTextStyle" => "true",
       "bigText" => "Air quality level of " . $level .
             " in " . $location . ", click to see more.",
       "lvl" => "0",
       "msg" => "There is no risks to your health."
 );
 $pbS->Payload($customfields);
$pbS->Alert("Air quality level of " . $level .
       " in " . $location . ", click to see more.");
$pbS->Platform(array(
       "0",
       "1"
```

```
        ));
        $pbS->Badge("+2");
        $pbS->Tags($tag);
        $pbS->Push();
        break;
case 1:
        $tag = array(
                $location . "Static",
                $location . "Walking",
                $location . "Sport",
                $location . "Transport"
        );
        // Big Text
        $customfields = array(
                "BigTextStyle" => "true",
                "bigText" => "Air quality level of " . $level . " in
                   " . $location . ", click to see more.",
                "lvl" => "1",
                "msg" => "Pollution and ozone can affect very
                        slightly unusually sensitive to respiratory
                        symptoms people.",
                "rec" => "Try not to make long effort or outdoor
                        activities soon."
                );
        $pbS->Payload($customfields);
        $pbS->Alert("Air quality level of " . $level .
                " in " . $location . ", click to see more.");
        $pbS->Platform(array(
                "0",
                "1"
        ));
        $pbS->Badge("+2");
        $pbS->Tags($tag);
        $pbS->Push();
        break;
case 2:
         $customfields = array(
                "BigTextStyle" => "true",
                "bigText" => "Air quality level of " . $level . " in
                   " . $location . ", click to see more.",
```

```
        "lvl" => "2",
        "msg" => "People with lung or heart disease,
               the elderly, and persons engaged in outdoor
               activities could be affected."
        );
$pbS->Payload($customfields);
$pbS->Alert("Air quality level of " . $level .
        " in " . $location . ", click to see more.");
$pbS->Platform(array(
        "0",
        "1"
));
$pbS->Badge("+2");
$pbS->Tags(array(
        $location . "Static",
        $location . "Transport"
));
$pbS->Push();
$customfields = array(
        "BigTextStyle" => "true",
        "bigText" => "Air quality level of " . $level . " in
           " . $location . ", click to see more.",
        "lvl" => "2",
        "msg" => "People with lung or heart disease,
                       the elderly, and persons engaged in
                          outdoor
                       activities could be affected.",
        "rec" => "Should reduce prolonged efforts
               outdoor."
);
$pbW->Payload($customfields);
$pbW->Alert("Air quality level of " . $level .
        " in " . $location . ", click to see more.");
$pbW->Platform(array(
        "0",
        "1"
));
$pbW->Badge("+2");
$pbW->Tags($location . "Walking");
$pbW->Push();
```

```
        $customfields = array(
               "BigTextStyle" => "true",
               "bigText" => "Air quality level of " . $level . " in
                  " . $location . ", click to see more.",
               "lvl" => "2",
               "msg" => "People with lung or heart disease,
                      the elderly, and persons engaged in outdoor
                      activities could be affected.",
               "rec" => "Should reduce intensive outdoor
                      efforts."
        );
        $pbR->Payload($customfields);
        $pbR->Alert("Air quality level of " . $level .
               " in " . $location . ", click to see more.");
        $pbR->Platform(array(
               "0",
               "1"
        ));
        $pbR->Badge("+2");
        $pbR->Tags($location . "Sport");
        $pbR->Push();
        break;
case 3:
        $customfields = array(
               "BigTextStyle" => "true",
               "bigText" => "Air quality level of " . $level . " in
                  " . $location . ", click to see more.",
               "lvl" => "3",
               "msg" => "Members of sensitive groups may
                      experience more serious health effects."
        );
        $pbS->Payload($customfields);
        $pbS->Alert("Air quality level of " . $level .
               " in " . $location . ", click to see more.");
        $pbS->Platform(array(
               "0",
               "1"
        ));
        $pbS->Badge("+2");
        $pbS->Tags(array(
```

```
        $location . "Static",
        $location . "Transport"
));
$pbS->Push();
$customfields = array(
        "BigTextStyle" => "true",
        "bigText" => "Air quality level of " . $level . " in
            " . $location . ", click to see more.",
        "lvl" => "3",
        "msg" => "Members of sensitive groups may experience
                more serious health effects.",
        "rec" => "Should avoid prolonged efforts outdoor.",
        "recGeneral" => "Should reduce prolonged outdoor
                efforts."
);
$pbW->Payload($customfields);
$pbW->Alert("Air quality level of " . $level . " in " .
        $location . ", click to see more.");
$pbW->Platform(array(
        "0",
        "1"
));
$pbW->Badge("+2");
$pbW->Tags($location . "Walking");
$pbW->Push();
$customfields = array(
        "BigTextStyle" => "true",
        "bigText" => "Air quality level of " . $level . " in
            " . $location . ", click to see more.",
        "lvl" => "3",
        "msg" => "Members of sensitive groups may experience
            more
                serious health effects.",
        "rec" => "Should avoid intensive outdoor efforts.",
        "recGeneral" => "Should reduce prolonged outdoor
            efforts."
);
$pbR->Payload($customfields);
$pbR->Alert("Air quality level of " . $level . " in " .
        $location . ", click to see more.");
```

```
        $pbR->Platform(array(
              "0",
              "1"
        ));
        $pbR->Badge("+2");
        $pbR->Tags($location . "Sport");
        $pbR->Push();
        break;
case 4:
        $customfields = array(
              "BigTextStyle" => "true",
              "bigText" => "Air quality level of " . $level . " in
                 " .
              $location . ", click to see more.",
              "lvl" => "4",
              "msg" => "Everyone can experience more serious health
                 effects.",
              "recGeneral" => "Should limit prolonged outdoor
                 efforts."
        );
        $pbS->Payload($customfields);
        $pbS->Alert("Air quality level of " . $level . " in " .
              $location . ", click to see more.");
        $pbS->Platform(array(
              "0",
              "1"
        ));
        $pbS->Badge("+2");
        $pbS->Tags(array(
              $location . "Static",
              $location . "Transport"
        ));
        $pbS->Push();
        $customfields = array(
              "BigTextStyle" => "true",
              "bigText" => "Air quality level of " . $level . " in
                 " . $location . ", click to see more.",
              "lvl" => "4",
              "msg" => "Everyone can experience more serious health
                 effects.",
```

```
                "rec" => "You must avoid any outside effort.",
                "recGeneral" => "Should reduce prolonged outdoor
                    efforts."
        );
        $pbW->Payload($customfields);
        $pbW->Alert("Air quality level of " . $level . " in " .
                $location . ", click to see more.");
        $pbW->Platform(array(
                "0",
                "1"
        ));
        $pbW->Badge("+2");
        $pbW->Tags($location . "Walking");
        $pbW->Push();
        $customfields = array(
                "BigTextStyle" => "true",
                "bigText" => "Air quality level of " . $level . " in
                    " . $location . ", click to see more.",
                "lvl" => "4",
                "msg" => "Everyone can experience more serious health
                    effects.",
                "rec" => "You must avoid any outside effort.",
                "recGeneral" => "Should reduce prolonged outdoor
                    efforts."
        );
        $pbR->Payload($customfields);
        $pbR->Alert("Air quality level of " . $level . " in " .
                $location . ", click to see more.");
        $pbR->Platform(array(
                "0",
                "1"
        ));
        $pbR->Badge("+2");
        $pbR->Tags($location . "Sport");
        $pbR->Push();
        break;
case 5:
        $tag = array(
                $location . "Estatico",
                $location . "Caminando",
```

```
                $location . "Esfuerzo",
                $location . "Transporte"
        );
        $customfields = array(
                "BigTextStyle" => "true",
                "bigText" => "Air quality level of " . $level . " in
                    " . $location . ", click to see more.",
                "lvl" => "5",
                "rec" => "You must avoid any outside effort.",
                "recGeneral" => "You must avoid any outside effort."
        );
        $pbS->Payload($customfields);
        $pbS->Alert("Air quality level of " . $level . " in " .
                $location . ", click to see more.");
        $pbS->Platform(array(
                "0",
                "1"
        ));
        $pbS->Badge("+2");
        $pbS->Tags($tag);
        $pbS->Push();
        break;
       default:
        break;
       }
  }
  else
  {
        echo "Alert already created.";
  }
});
$app->run();
?>
```

D.1.2. GetAlertsByLocation

```
<?php
require 'Slim/Slim.php';
SlimSlim::registerAutoloader();
$app = new SlimSlim();
$app->get('/alerts/:location',
```

```
function ($location)
 {
  include_once "conection.php"; // File with credentials
  $sql = "SELECT id, level, location, IDtimestamp
              FROM alert WHERE location LIKE '" .
              $location . "' ORDER BY IDtimestamp DESC LIMIT 5 ";
  $result = mysql_query($sql);
  $Id = array();
  $Level = array();
  $Location = array();
  $IDtimestamp = array();
  $cant = mysql_num_rows($result);
  if ($cant != 0)
  {
        $i = 0;
        while ($row = mysql_fetch_row($result))
        {
              $Id[$i] = $row[0];
              $Level[$i] = $row[1];
              $Location[$i] = $row[2];
              $IDtimestamp[$i] = $row[3];
              $i++;
        }
  }
  $i = 0;
  $named_array = array(
              'alerts' => array()
  );
  while ($i < $cant)
  {
        $named_array['alerts'][$i] = array(
              "Id" => $Id[$i],
              "Level" => $Level[$i],
              "Location" => $Location[$i],
              "IDtimestamp" => $IDtimestamp[$i]
        );
        $i++;
  }
  if ($cant != 0)
        echo json_encode($named_array); //Code it in JSON and return.
```

```
  else
        echo 0; //No alerts.
});
$app->run();
?>
```

D.2. Android

D.2.1. FragmentGPS

```
public class FragmentGPS extends Fragment {
 int cont = 0; static String defLocation = "Cadiz", location;
 private class sendContext extends AsyncTask < Void, Void, String > {
  double latitude,
  longitude,
  speed;
  DBTag MDB;
  DBSuggestions DBS;

  public sendContext(double latitude, double longitude, double
     speed) {
   this.latitude = latitude;
   this.longitude = longitude;
   this.speed = speed;
   MDB = new DBTag(getActivity());
   DBS = new DBSuggestions(getActivity());
  }

  public String stripAccents(String s) {
   s = Normalizer.normalize(s, Normalizer.Form.NFD);
   s = s.replaceAll("[\\p{InCombiningDiacriticalMarks}]", "");
   return s;
  }

  @Override
  protected String doInBackground(Void...params) {
   /** GET THE ADMIN AREA (PROVINCE/STATE) FROM LAT & LONG **/
   Geocoder geoCoder = new Geocoder(getActivity().getBaseContext(),
        Locale.getDefault());
   try {
    List < Address > addresses =
```

```
                geoCoder.getFromLocation(latitude, longitude, 1);
    if (addresses.size() > 0) {
     if (addresses.get(0).getSubAdminArea() != null)
      return stripAccents(addresses.get(0).getSubAdminArea());
     else
      return stripAccents(addresses.get(0).getLocality());
    }
   } catch (IOException e1) {
    e1.printStackTrace();
   }
   return "null";
  }

  protected void onPostExecute(String loc) {
   /** GET LOCATION **/
   if (loc.equals("null"))
    location = defLocation;
   else
    location = loc;
   /** GENERAL VARIABLES **/
   TextView speedT = (TextView) getView().findViewById(R.id.speed);
   String lastTag = MDB.getTag();
   /** GET ACTIVITY **/
   if (!lastTag.equals("noAlerts")) {
    String newTag = location;
    if (speed < 1) {
     String aux = speedT.getText().toString();
     newTag += "Static";
     aux += "\n Static in " + location + "\n LT: " + lastTag +
               " | NT: " + newTag;
     speedT.setText(aux);
    } else if (speed >= 1 && speed <= 2.7) {
     String aux = speedT.getText().toString();
     newTag += "Walking";
     aux += "\n Walking in " + location + "\n LT: " + lastTag +
               " | NT: " + newTag;
     speedT.setText(aux);
    } else if (speed > 2.7 && speed < 5.5) {
     String aux = speedT.getText().toString();
     newTag += "Sport";
```

```
 aux += "\n Physical activity in " + location + "\n LT: " +
    lastTag +
           " | NT: " + newTag;
 speedT.setText(aux);
} else if (speed >= 5.5) {
 String aux = speedT.getText().toString();
 newTag += "Transport";
 aux += "\n Transport in " + location + "\n LT: " + lastTag +
          " | NT: " + newTag;
 speedT.setText(aux);
} else {
 String aux = speedT.getText().toString();
 aux += "\n OUT OF RANGE";
 speedT.setText(aux);
}
//Now we check if the tag changes or not
lastTag = MDB.getTag();
if (!lastTag.equals(newTag)) {
 MDB.deleteTag();
 MDB.setTag(newTag);
 Pushbots.sharedInstance().untag(lastTag);
 Pushbots.sharedInstance().tag(newTag);
 Log.w("MainActivity", "User tag updated.");
} else {
 Log.w("MainActivity", "User tag not changes.");
}
} else {
 Log.w("MainActivity",
     "Alerts of quality level off, so we don't change the tag.");
 String aux = speedT.getText().toString();
 aux += "\n Alerts are off.";
 speedT.setText(aux);
}
String suggestions = DBS.getSug();
if (!suggestions.equals("noSug") && !suggestions.equals(location
   + "Sug")) {
 DBS.deleteSug();
 DBS.setSug(location + "Sug");
 Pushbots.sharedInstance().untag(suggestions);
 Pushbots.sharedInstance().tag(location + "Sug");
```

```
    Log.w("MainActivity", "Suggestions are updated.");
   } else {
    Log.w("MainActivity",
         "Suggestions are OFF or the user is in the same location.");
   }
  }
 }

 @Override
 public View onCreateView(LayoutInflater inflater, ViewGroup
     container,
  Bundle savedInstanceState) {
  final View view = inflater.inflate(R.layout.activity_gps,
     container,
   false);
  LocationManager locationManager = (LocationManager) getActivity()
   .getSystemService(Context.LOCATION_SERVICE);
  // Check if GPS is enabled or not
  if
     (!locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER))
   Toast.makeText(getActivity(), "Turn on the GPS sensor please.",
       Toast.LENGTH_LONG).show();
  // Define a listener that responds to location updates
  LocationListener locationListener = new LocationListener() {
   public void onLocationChanged(Location location) {
    location.getLatitude();
    TextView speed = (TextView) view.findViewById(R.id.speed);
    String d = new SimpleDateFormat("dd-MM-yyyy
       HH:mm:ss.S").format(new Date());
    if (location.hasSpeed()) {
     String aux = speed.getText().toString();
     aux += "\nCont = " + cont + " | Current speed: " +
        location.getSpeed();
     speed.setText(aux);
    } else {
     String aux = speed.getText().toString();
     aux += "\nCont = " + cont + " | No speed available";
     speed.setText(aux);
    }
    TextView loc = (TextView) view.findViewById(R.id.loc);
```

```
    String aux = loc.getText().toString();
    aux += "\nCont = " + cont + " | Date = " + d + "\n| Lat: " +
              location.getLatitude() + "\nLong: " +
                 location.getLongitude();
    loc.setText(aux);
    new sendContext(location.getLatitude(), location.getLongitude(),
         location.getSpeed()).execute();
    cont++;
   }

   public void onStatusChanged(String provider, int status,
    Bundle extras) {}

   public void onProviderEnabled(String provider) {}

   public void onProviderDisabled(String provider) {}

  };
  locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
     15000,
   100, locationListener);
  return view;
 }
}
```

D.2.2. AlarmReceiver

```
public class AlarmReceiver extends BroadcastReceiver {
 int dayOfWeek;

 @Override
 public void onReceive(Context context, Intent intent) {
  //Start the object PB
  Pushbots.sharedInstance().init(context);
  Log.d("AlarmReceiver", "Alarm catched!");
  Calendar c = Calendar.getInstance();
  c.setTime(new Date());
  dayOfWeek = c.get(Calendar.DAY_OF_WEEK) - 1;
  Calendar rightNow = Calendar.getInstance();
  int hour = rightNow.get(Calendar.HOUR_OF_DAY);
  if (setContext(hour, context)) {
```

```
/*CANCEL THIS ALARM*/
Intent alarmIntentC = new Intent(context, AlarmReceiver.class);
PendingIntent pendingIntentC =
     PendingIntent.getBroadcast(context, 0, alarmIntentC, 0);
AlarmManager managerC =
     (AlarmManager)
        context.getSystemService(Context.ALARM_SERVICE);
managerC.cancel(pendingIntentC);
/*SET THE NEW ONE*/
int nextHour = -1;
if (hour >= 0 && hour < 8) {
 nextHour = 8;
} else if (hour >= 8 && hour < 14) {
 nextHour = 14;
} else if (hour >= 14 && hour < 16) {
 nextHour = 16;
} else if (hour >= 16 && hour < 20) {
 nextHour = 20;
} else if (hour >= 20 && hour < 24) {
 nextHour = 0;
}
Intent alarmIntent = new Intent(context, AlarmReceiver.class);
PendingIntent pendingIntent =
     PendingIntent.getBroadcast(context, 0, alarmIntent, 0);
AlarmManager manager =
     (AlarmManager)
        context.getSystemService(Context.ALARM_SERVICE);
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.set(Calendar.HOUR_OF_DAY, nextHour);
calendar.set(Calendar.MINUTE, 1);
if (nextHour == 0)
 calendar.add(Calendar.DAY_OF_MONTH, 1);
manager.set(AlarmManager.RTC_WAKEUP,
     calendar.getTimeInMillis(), pendingIntent);
Log.d("AlarmReceiver",
     "New Alarm Set for " + calendar.toString());
} else {
 Log.d("AlarmReceiver","As you dont have any activity in your
    timetable,
```

```
      we dont set the alarm.");
 }
}

public boolean setContext(int hour, Context ctx) {
 boolean flag = true;
 int activity = -1;
 DBTimetable DBT = new DBTimetable(ctx);
 if (hour >= 0 && hour < 8) {
  activity = DBT.getActivity(dayOfWeek, 1);
 } else if (hour >= 8 && hour < 14) {
  activity = DBT.getActivity(dayOfWeek, 2);
 } else if (hour >= 14 && hour < 16) {
  activity = DBT.getActivity(dayOfWeek, 3);
 } else if (hour >= 16 && hour < 20) {
  activity = DBT.getActivity(dayOfWeek, 4);
 } else if (hour >= 20 && hour < 24) {
  activity = DBT.getActivity(dayOfWeek, 5);
 }
 String location = "";
 String TAG = "";
 switch (DBT.getState(dayOfWeek)) {
  case 1:
   location = "Almeria";
   break;
  case 2:
   location = "Cadiz";
   break;
  case 3:
   location = "Cordoba";
   break;
  case 4:
   location = "Granada";
   break;
  case 5:
   location = "Huelva";
   break;
  case 6:
   location = "Jaen";
   break;
```

```
   case 7:
    location = "Malaga";
    break;
   case 8:
    location = "Sevilla";
    break;
  }
  switch (activity) {
   case 1:
    TAG = location + "Static";
    break;
   case 2:
    TAG = location + "Walking";
    break;
   case 3:
    TAG = location + "Sport";
    break;
   case 4:
    TAG = location + "Transport";
    break;
  }
  DBTag MDB = new DBTag(ctx);
  //Now we check if the tag changes or not
  String lastTag = MDB.getTag();
  /** GET ACTIVITY **/
  if (!lastTag.equals("noAlerts")) {
   if (!lastTag.equals(TAG)) {
    MDB.deleteTag();
    MDB.setTag(TAG);
    Pushbots.sharedInstance().untag(lastTag);
    if (TAG.equals("")) {
     Log.w("AlaramReceiver",
          "No activity for this period, so dont put a TAG Alert.");
     flag = false;
    } else {
     Pushbots.sharedInstance().tag(TAG);
     flag = true;
    }
    Log.w("AlaramReceiver", "User tag updated.");
   } else {
```

```
    Log.w("AlaramReceiver", "User tag not changes.");
   }
  } else {
   Log.w("AlaramReceiver",
        "Alerts of quality level off, so we don't change the tag.");
  }
  DBSuggestions DBS = new DBSuggestions(ctx);
  String suggestions = DBS.getSug();
  if (!suggestions.equals("noSug") && !suggestions.equals(location +
     "Sug")) {
   DBS.deleteSug();
   DBS.setSug(location + "Sug");
   Pushbots.sharedInstance().untag(suggestions);
   if (location.equals("")) {
    Log.w("AlaramReceiver",
         "No activity for this period, so dont put a TAG Sug.");
    flag = false;
   } else {
    Pushbots.sharedInstance().tag(location + "Sug");
    flag = true;
   }
   Log.w("AlaramReceiver", "Suggestions are updated.");
  } else {
   Log.w("AlaramReceiver",
        "Suggestions are OFF or the user is in the same location.");
  }
  return flag;
 }
}
```

D.3. Consumidor de eventos

```
#!/usr/bin/python
# -*- coding: latin-1 -*-
# Copyright 2013 Telefonica Investigacion y Desarrollo, S.A.U
#
# This file is part of Orion Context Broker.
#
# Orion Context Broker is free software: you can redistribute it
   and/or
# modify it under the terms of the GNU Affero General Public License
```

```
   as
# published by the Free Software Foundation, either version 3 of the
# License, or (at your option) any later version.
#
# Orion Context Broker is distributed in the hope that it will be
   useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
   Affero
# General Public License for more details.
#
# You should have received a copy of the GNU Affero General Public
   License
# along with Orion Context Broker. If not, see
   http://www.gnu.org/licenses/.
#
# For those usages not covered by this license please contact with
# iot_support at tid dot es
from __future__ import division # need for seconds calculation
   (could be removed with Python 2.7)
__author__ = 'fermin'
# This program stores everything it receives by HTTP in a given URL
   (pased as argument),
# Then return the accumulated data upon receiving 'GET
   <host>:<port>/dump'. It is aimet
# at harness test for subscription scenarios (so
   accumulator-server.py plays the role
# of a subscribed application)
#
# Known issues:
#
# * Curl users: use -H "Expect:" (default "Expect: 100-continue"
   used by curl has been problematic
# in the past)
# * Curl users: use -H "Content-Type: application/xml" for XML
   payload (the default:
# "Content-Type: application/x-www-form-urlencoded" has been
   problematic in the pass)
from flask import Flask, request, Response
from sys import argv, exit
```

```
from datetime import datetime
from math import trunc
from time import sleep
import os
import atexit
import string
import signal
import json
# This function is registered to be called upon termination
def all_done():
    os.unlink(pidfile)
# Default arguments
port = 1028
host='0.0.0.0'
server_url = '/accumulate'
verbose = 0
# Arguments from command line
if len(argv) > 2:
    port = int(argv[1])
    server_url = argv[2]
if len(argv) > 3:
    if argv[3] == 'on':
        print 'verbose mode is on'
        verbose = 1
    else:
        host = argv[3]
if len(argv) > 4:
    if argv[4] == 'on':
        print 'verbose mode is on'
        verbose = 1
pid = str(os.getpid())
pidfile = "/tmp/accumulator." + str(port) + ".pid"

#
# If an accumulator process is already running, it is killed.
# First using SIGTERM, then SIGINT and finally SIGKILL
# The exception handling is needed as this process dies in case
# a kill is issued on a non-running process ...
#
if os.path.isfile(pidfile):
```

```
    oldpid = file(pidfile, 'r').read()
    opid = string.atoi(oldpid)
    print "PID file %s already exists, killing the process %s" %
        (pidfile, oldpid)

    try:
        oldstderr = sys.stderr
        sys.stderr = open("/dev/null", "w")
        os.kill(opid, signal.SIGTERM);
        sleep(0.1)
        os.kill(opid, signal.SIGINT);
        sleep(0.1)
        os.kill(opid, signal.SIGKILL);
       sys.stderr = oldstderr except:
        print "Process %d killed" % opid
#
# Creating the pidfile of the currently running process
#
file(pidfile, 'w').write(pid)

#
# Making the function all_done being executed on exit of this
   process.
# all_done removes the pidfile # atexit.register(all_done)
app = Flask(__name__)
@app.route("/noresponse", methods=['POST'])
def noresponse():
    sleep(10)
    return Response(status=200)

@app.route("/noresponse/updateContext", methods=['POST'])
def unoresponse():
    sleep(10)
    return Response(status=200)

@app.route("/noresponse/queryContext", methods=['POST'])
def qnoresponse():
    sleep(10)
    return Response(status=200)
```

```
@app.route("/v1/updateContext", methods=['POST'])
@app.route("/v1/queryContext", methods=['POST'])
@app.route(server_url, methods=['GET', 'POST', 'PUT', 'DELETE'])
def record():
    global ac, t0, times
    s = '' valueJSON = ''
    send_continue = False

    # First notification? Then, set reference datetime. Otherwise,
       add the
    # timedelta to the list
    if (t0 == ''):
        t0 = datetime.now()
        times.append(0)
    else:
        delta = datetime.now() - t0
        # Python 2.7 could use delta.total_seconds(), but we use
           this formula
        # for backward compatibility with Python 2.6
        t = (delta.microseconds + (delta.seconds + delta.days * 24 *
           3600) * 10**6) / 10**6
        times.append(trunc(round(t)))
       #times.append(t)

    # Store verb and URL
    s += request.method + ' ' + request.url + '\n'
    # Store headers
    for h in request.headers.keys():
        s += h + ': ' + request.headers[h] + '\n'
        if ((h == 'Expect') and (request.headers[h] ==
           '100-continue')):
            send_continue = True
    # Store payload
    if ((request.data is not None) and (len(request.data) != 0)):
        s += '\n'
        s += request.data
        valueJSON += request.data
    # Separator
    s += '========================================\n'
    # Accumulate
```

```
    ac += s

        if verbose:
        print s
        valueJSON = valueJSON.replace('\n','')
        valueJSON = valueJSON.replace(' ','')
        import json
        j = json.loads(valueJSON)
        print "Nivel de alarma %s" %
            j['contextResponses'][0]['contextElement']
                        ['attributes'][0]['value']
        print "Localidad %s" %
            j['contextResponses'][0]['contextElement']
                        ['attributes'][1]['value']
        print "Timestamp %s" %
            j['contextResponses'][0]['contextElement']
                        ['attributes'][2]['value']
        nivel =
            j['contextResponses'][0]['contextElement']['attributes']
                        [0]['value']
        localidad =
            j['contextResponses'][0]['contextElement']['attributes']
                        [1]['value']
            idtime =
                j['contextResponses'][0]['contextElement']['attributes']
                        [2]['value']
        url =
            "http://130.206.117.253/SlimWS/app/alert.php/addAlert/%i/%s/%s"
                        % (int(nivel),localidad,idtime)
        print "Realizando peticion a %s" % url
        import urllib2
        urllib2.urlopen(url).read()
        print "Notificacion creada"

        if send_continue:
                return Response(status=100)
    else:
        return Response(status=200)

@app.route('/dump', methods=['GET'])
```

```
def dump():
    return ac


@app.route('/times', methods=['GET'])
def times():
   return ', '.join(map(str,times)) + '\n'


@app.route('/number', methods=['GET'])
def number():
    return str(len(times)) + '\n'


@app.route('/reset', methods=['POST'])
def reset():
    global ac, t0, times
    ac = ''
    t0 = ''
    times = []
    return Response(status=200)


@app.route('/pid', methods=['GET'])
def getPid():
    return str(os.getpid())

# This is the accumulation string
ac = '' t0 = '' times = []
if __name__ == '__main__':
    # Note that using debug=True breaks the the procedure to write
       the PID into a file. In particular
    # makes the calle os.path.isfile(pidfile) return True, even if
       the file doesn't exist. Thus,
    # use debug=True below with care :)
    app.run(host=host, port=port)
```

Bibliografía

[1] Air Quality Index - EEUU (Mayo 2016), https://www3.epa.gov/airnow/aqi_brochure_02_14.pdf

[2] Amazon Web Services (Abril 2016), https://aws.amazon.com/es/

[3] Android (Mayo 2016), https://www.android.com/intl/es_es/

[4] Android Studio (Mayo 2016), https://developer.android.com/studio/intro/index.html?hl=es

[5] Apache Flink (Mayo 2016), https://flink.apache.org/

[6] Apache Maven (Mayo 2016), http://maven.apache.org/

[7] Bases de datos MySQL (Mayo 2016), https://www.oracle.com/es/products/mysql/overview/index.html

[8] Business event processing (BEP) (Mayo 2016), http://www.ibm.com/support/knowledgecenter/SSTNLG_6.2.1/com.ibm.wbe.overview.doc/doc/businesseventprocessing.html

[9] Calidad del aire (exterior) y salud (Junio 2016), http://www.who.int/mediacentre/factsheets/fs313/es/

[10] ¿Computación en la Nube? que es y como funciona... (Mayo 2016), http://tecnato.com/computacion-en-la-nube-que-es-y-como-funciona/

[11] Conceptos sobre APIs REST (Mayo 2016), http://asiermarques.com/2013/conceptos-sobre-apis-rest/

[12] Consumidor de eventos Python (Mayo 2016), https://github.com/telefonicaid/fiware-orion/blob/master/scripts/accumulator-server.py

[13] Diagrama de Gantt (Mayo 2016), http://es.ccm.net/contents/580-diagrama-de-gantt

[14] Documentación ListView (Mayo 2016), https://developer.android.com/guide/topics/ui/layout/listview.html?hl=es

[15] EventZero (Mayo 2016), https://www.eventzero.com/

[16] EPA (Mayo 2016), https://www3.epa.gov/

[17] Esper EPL Online (Mayo 2016), http://es.ccm.net/contents/580-diagrama-de-gantt

[18] Esper: Event Processing (Mayo 2016), http://www.espertech.com/products/esper.php

[19] Estándar ISO 8601 (Mayo 2016), http://www.iso.org/iso/home/standards/iso8601.htm

[20] FI-WARE NGSI-10 Open RESTful API Specification (Mayo 2016), https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-10_Open_RESTful_API_Specification

[21] Fiware CEPHEUS enabler (Mayo 2016), https://fiware-cepheus.readthedocs.io/en/latest/index.html

[22] FIWARE Cosmos enabler (Mayo 2016), https://fiware-cosmos.readthedocs.io/en/latest/

[23] Fiware ORION enabler (Mayo 2016), http://fiware-orion.readthedocs.io/en/latest/index.html

[24] Future Internet WARE (Abril 2016), https://www.fiware.org/

[25] GanttProject (Mayo 2016), http://www.ganttproject.biz/

[26] Gartner Says Worldwide Smartphone Sales Grew 9.7 Percent in Fourth Quarter of 2015 (Junio 2016), http://www.gartner.com/newsroom/id/3215217

[27] GNU General Public License (Mayo 2016), http://www.gnu.org/copyleft/gpl.html

[28] Google App Engine Documentation (Abril 2016), https://cloud.google.com/appengine/docs

[29] Google Cloud Messaging (Mayo 2016), https://developers.google.com/cloud-messaging/

[30] Grupo UCASE de Ingeniería del Software (Enero 2016), https://ucase.uca.es/

[31] HTTP Method Definitions (Mayo 2016), https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

[32] IBM ISII Complex Event Processing: De la detección de eventos a la acción inmediata (Abril 2016), https://www.ibm.com/developerworks/ssa/local/im/ssa/identity-insight-complex-event-processing/

[33] Impactos sobre la salud de la contaminación atmosférica (Junio 2016), http://ecodes.org/salud-calidad-aire/201302176117/Impactos-sobre-la-salud-de-la-contaminacion-atmosferica

[34] Instalar Cepheus (Mayo 2016), http://fiware-cepheus.readthedocs.io/en/latest/admin/cep/index.html

[35] Instalar Orion Context Broker (Mayo 2016), https://fiware-orion.readthedocs.io/en/develop/admin/build_source/index.html

[36] IntellJ IDEA (Mayo 2016), https://www.jetbrains.com/idea/features/

[37] Internet de las cosas (Mayo 2016), http://hipertextual.com/2015/06/internet-of-things

[38] Introducción al procesamiento de Eventos Complejos (Abril 2016), http://www.decidesoluciones.es/introduccion-al-procesamiento-de-eventos-complejos-ii/

[39] iOS (Mayo 2016), http://www.apple.com/es/ios/what-is/

[40] Junta de Andalucía: Calidad del aire. (Junio 2016), http://www.juntadeandalucia.es/temas/medio-ambiente/emisiones/calidad.html

[41] La contaminación atmosférica aumentó en 2015 (Junio 2016), http://www.efe.com/efe/espana/sociedad/la-contaminacion-atmosferica-aumento-en-2015-tras-siete-anos-de-descensos/10004-2948475#

[42] Licencia Apache 2.0 (Mayo 2016), http://www.apache.org/licenses/LICENSE-2.0

[43] Microsoft StreamInsight (Mayo 2016), https://technet.microsoft.com/es-es/library/ee362541(v=sql.111).aspx

[44] National Air Quality Index - India (Mayo 2016), http://pib.nic.in/newsite/PrintRelease.aspx?relid=110654

[45] Índice de Calidad del Aire - India (Mayo 2016), http://www.bsc.es/projects/earthscience/visor/bases_datos/image_viewer/docs/Descripcion_ICA_y_NCA.pdf

[46] Neptuno Redmine (Mayo 2016), https://neptuno.uca.es/redmine/projects

[47] OMS (Mayo 2016), http://www.who.int/es/

[48] Oracle CEP 10g (Mayo 2016), http://docs.oracle.com/cd/E13157_01/wlevs/docs30/

[49] Payload (Mayo 2016), http://www.pcmag.com/encyclopedia/term/48909/payload

[50] phpMyAdmin (Mayo 2016), https://www.phpmyadmin.net/

[51] phpMyAdmin documentación (Junio 2016), https://www.phpmyadmin.net/docs/

[52] Primeros pasos en bases de datos Android (Mayo 2016), http://www.sgoliver.net/blog/bases-de-datos-en-android-i-primeros-pasos/

[53] Procesador de eventos complejos WSO2 (Junio 2016), http://wso2.com/products/complex-event-processor/

[54] Procesamiento de Eventos Complejos en Entornos SOA: Caso de Estudio para la Detección Temprana de Epidemias (Junio 2016), http://lbd.udc.es/jornadas2011/actas/JCIS/JCIS/S3/S3_1_paper.pdf

[55] Protocolo SHH (Mayo 2016), http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html

[56] PushBots (Mayo 2016), http://pushbots.com/

[57] ¿Qué es Big Data? (Mayo 2016), https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/

[58] ¿Qué es IaaS? (Abril 2016), http://www.interoute.es/what-iaas

[59] ¿Qué es PaaS? (Abril 2016), http://www.interoute.es/what-paas

[60] ¿Qué es SaaS? (Abril 2016), http://www.interoute.es/what-saas

[61] ¿Qué es un 'framework'? (Mayo 2016), http://jordisan.net/blog/2006/que-es-un-framework/

[62] ¿Qué es y para qué sirve JSON? (Mayo 2016), https://geekytheory.com/json-i-que-es-y-para-que-sirve-json/

[63] Slim PHP Framework (Mayo 2016), http://www.slimframework.com/

[64] StreamBase (Mayo 2016), http://www.streambase.com/

[65] Sublime Text (Mayo 2016), https://www.sublimetext.com/

[66] Terminator (Mayo 2016), https://apps.ubuntu.com/cat/applications/precise/terminator/

[67] Triceps CEP (Mayo 2016), http://triceps.sourceforge.net/docs-latest/guide.html

[68] Un Estudio sobre el Procesamiento de Eventos Complejos y su Integración en Arquitecturas Orientadas a Servicios (Junio 2016), https://ucase.uca.es/sites/default/files/publications/jboubeta/2010-Ponencia-Boubeta-II-JORPRESI.pdf

[69] What is GPS? (Mayo 2016), http://www8.garmin.com/aboutGPS/

[70] Boubeta-Puig, J.: Desarrollo Dirigido por Modelos de Interfaces Específicas de Dominio para el Procesamiento de Eventos Complejos en Arquitecturas Orientadas a Servicios. Ph.D. thesis, Universidad de Cádiz, Cádiz, España (Julio 2014), http://rodin.uca.es:80/xmlui/handle/10498/17554

[71] Boubeta-Puig, J., Ortiz, G., Medina-Bulo, I.: Una Propuesta Orientada a Servicios para la Prevención de Riesgos Personales Derivados de la Calidad del Aire. In: Actas de las X Jornadas de Ciencia e Ingeniería de Servicios. pp. 58–67. Cádiz, España (September 2014), http://sistedes2014.uca.es/Actas-JCIS-2014.pdf

[72] Boubeta-Puig, J., Ortiz, G., Medina-Bulo, I.: MEdit4CEP: A model-driven solution for real-time decision making in SOA 2.0. Knowledge-Based Systems 89, 97–112 (Noviembre 2015), http://www.sciencedirect.com/science/article/pii/S0950705115002397

[73] Boubeta-Puig, J., Ortiz, G., Medina-Bulo, I.: ModeL4CEP: Graphical domain-specific modeling languages for CEP domains and event patterns. Expert Systems with Applications 42(21), 8095–8110 (Noviembre 2015), http://www.sciencedirect.com/science/article/pii/S0957417415004479

[74] Etzion, O., Niblett, P.: Event Processing in Action. Manning, Stamford, USA (2011), http://www.manning.com/etzion/

[75] Luckham, D.: Event Processing for Business: Organizing the Real-Time Enterprise. Wiley, New Jersey, USA (2012), http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470534850.html