UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

Escuela Técnica Superior de Ingenieros de Telecomunicación

Universidad Politécnica de Valencia

GeoRBAC: Diseño e implementación del procesado de eventos de control de acceso

Proyecto Final de Carrera

Ingeniería Superior de Telecomunicaciones

Autor: Eneko Olivares Gorriti

Director: Carlos Enrique Palau Salvador

8 de Diciembre de 2014

A mis amigos por ayudarme siempre que han podido, en especial a Ignacio y Alejandro, con quienes he compartido mucho tiempo trabajando.

Al director de este proyecto, Carlos Palau, porque me ha brindado la posibilidad de realizar este proyecto y ha estado siempre dispuesto a echarme una mano.

A mis padres y a mis hermanas, por su apoyo incondicional y por haber cuidado de mí todo este tiempo.

Resumen

Este documento es la memoria de un proyecto nace a partir de un trabajo realizado de forma conjunta con otros dos compañeros dentro de la asignatura Redes Corporativas 2. En él se aprendió a utilizar un procesador de eventos complejo para aplicarlo en situaciones muy sencillas de control de acceso, en un entorno simulado.

Como continuación del trabajo realizado, se ampliará a un sistema más complejo, integrado en un entorno de sensores móviles, con la intención de poder ser aplicado en una situación real.

El sistema se dividió en tres partes, y este proyecto se centrará en el desarrollo de la aplicación que realizará la función de procesador de eventos dentro del sistema GEO-RBAC. El proyecto se ha estructurado en ocho capítulos.

El primer capítulo trata de sentar la base sobre la que se desarrollará el resto del proyecto. Se expone la situación actual de los sistemas RBAC para el control de acceso y de los estándares geoespaciales que existen, también se comentarán la motivación y objetivos que justifican el desarrollo de este proyecto. También se establecen los requerimientos que debería cumplir la aplicación diseñada.

El segundo, tercer y cuarto capítulo se centran en el desarrollo de la aplicación. En el segundo capítulo se listan las herramientas, tecnologías y servicios existentes con los cuales se construirá la aplicación, en el tercer capítulo se expondrá la arquitectura del sistema completo y del procesador de eventos. En el cuarto capítulo se mostrará el funcionamiento y la estructura de la aplicación desarrollada.

Por último, los tres últimos capítulos se enfocan en la puesta en marcha del sistema. En el quinto capítulo se mostrarán las aplicaciones auxiliares que se han desarrollado para este proyecto, en el sexto se realizarán las pruebas de ejecución necesarias y se recopilan los resultados obtenidos. Finalmente, en el séptimo y último capítulo se concluye el proyecto interpretando los resultados obtenidos y listando una serie de mejoras posibles para realizar en un futuro.

Índice general

Capítulo 1

Introducción

Las técnicas de control de acceso, ya sea en entornos físicos o lógicos, son necesarias para garantizar un nivel mínimo de seguridad en el sistema. De entre los distintos mecanismos para realizar el control de acceso, una de las técnicas más utilizadas es el control de acceso basado en roles, o RBAC.

Es posible extender RBAC para que tome en consideración otros atributos del usuario aparte de su rol. En aplicaciones que tengan una representación espacial, uno de los atributos principales de los usuarios es su posición. GEO-RBAC extiende RBAC para que la aplicación sea consciente de la posición del usuario a la hora de realizar el control de acceso.

En este proyecto se desarrollará una aplicación que se integra en un sistema GEO-RBAC para realizar el control de acceso en un entorno físico. El objetivo es entender el funcionamiento interno y sentar la base para un desarrollo posterior más completo, por tanto, el ámbito de aplicación del sistema estará acotado a un caso de uso concreto para que en ciclos posteriores pueda ser ampliado.

Además, se hará uso del servicio web SOS (Sensor Observation Service) para la recopilación de las posiciones de los dispositivos, por tanto, se describirá y desarrollará la integración de este servicio en un sistema de control de acceso GEO-RBAC.

1.1. Estado del arte

En esta sección comentaremos la situación actual de las tecnologías que vamos a aplicar en el desarrollo de nuestro sistema de control de acceso: el

servicio web SOS y los diferentes modelos de control de acceso basados en roles.

1.1.1. OGC y el estándar Sensor Observation Service

Los orígenes de OGC (Open Geospatial Consortium) se remontan a la década de los 80, cuando se desarrolla GRASS (Geographic Resources Analysis Support System) por el Cuerpo de Ingenieros del Ejército de Estados Unidos, que en poco tiempo se expandió entre las universidades gracias a que Internet empezó a expandirse. De esta forma se convirtió en uno de los primeros proyectos Open Source.

Sin embargo, en la década de los 90 el proyecto se tuvo que trasladar al sector privado por otras consideraciones. Es entonces cuando se funda OFG (Open GRASS Foundation) con el fin de emular este sistema pero con la misma filosofía Open Source. Pero pronto se establecieron nuevos objetivos, y el proyecto evolucionó para incorporar nuevas funcionalidades e intentar crear una solución completamente interoperable a través de interfaces abiertas en la red, pasándose a llamar por unos años OpenGIS Project hasta que se consolidó como Open Geospatial Consortium y que cuenta actualmente con 507 compañías, agencias gubernamentales y universidades. [1]

OGC ha desarrollado todo un ecosistema de soluciones interoperables para proveer de información geoespacial a Internet. En la siguiente figura se muestran los estándares más importantes desarrollados por OGC:

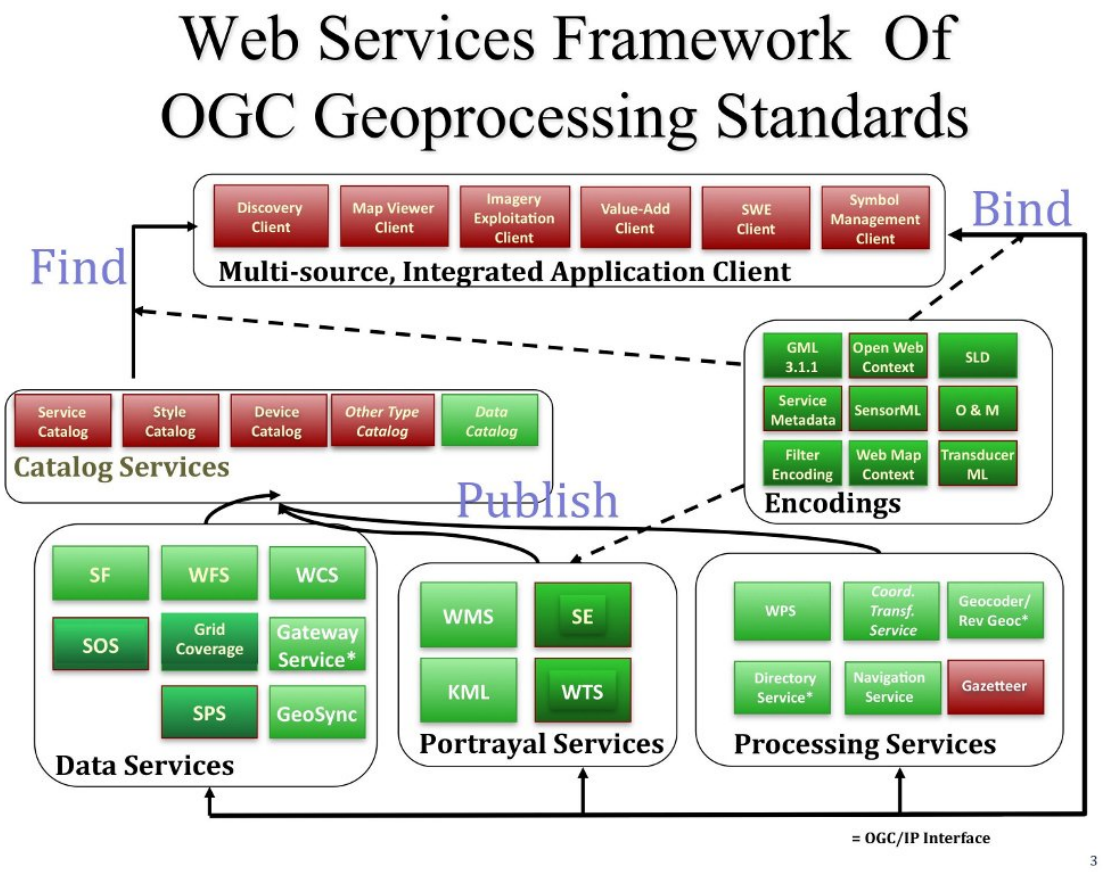


Figura 1.1: Estándares principales desarrollados por OGC [2]

En este proyecto utilizaremos el servicio web SOS, que forma parte del grupo Sensor Web Enablement, cuya función es estandarizar de qué forma son descubiertos los sensores y cómo acceder a ellos mediante la Web. Actualmente, SWE ha desarrollado los siguientes estándares: [2] [3]

- **Observations & Measurements (O&M):** Modelos estándar y esquema XML para codificar observaciones y medidas de un sensor, ya estén guardados o sean en tiempo real.

- **Sensor Model Language (SensorML):** Modelos estándar y esquema XML para describir sistemas de sensores y procesos asociados con observaciones de sensores. Provee de la información necesaria para descubrir sensores, localizar observaciones de sensores y procesarlos a bajo nivel.

- **Transducer Model Language (TransducerML or TML):** Es el modelo conceptual y esquema XML para describir transductores y dar soporte al flujo de datos en tiempo real desde y hacia sistemas de sensores.

- **Sensor Observations Service (SOS):** Define la interfaz web estándar para ejecutar peticiones, filtrar y recuperar observaciones e información de sistemas de sensores. Es el intermediario entre un cliente y un repositorio de observaciones o un canal en tiempo real.

- **Sensor Planning Service (SPS):** Define la interfaz web estándar para ejecutar peticiones de adquisiciones y observaciones realizadas por parte del usuario. Es el intermediario entre el cliente y un entorno de gestión de colecciones de sensores.

- **Sensor Alert Service (SAS):** Define la interfaz web estándar para publicar y subscribirse a alertas de sensores.

- **Web Notification Services (WNS):** Define la interfaz web estándar para realizar la entrega de mensajes o alertas de forma asíncrona por parte de servicios web SAS y SPS.

En abril de 2012 se adoptó la segunda versión del estándar que define el servicio web SOS, cambio necesario para reestructurar la especificación separando el núcleo de sus extensiones. Entre otras mejoras, se incrementó la interoperabilidad del servicio añadiendo soporte para otros formatos de intercambio de mensajes y creando nuevas operaciones para realizar filtrado temporal y espacial. También se rediseñó la forma de gestionar los resultados. [4]

OGC sólo define la interfaz que debe cumplir un servicio web SOS, no su implementación. Actualmente existen diferentes implementaciones, de entre las cuales cabe destacar IstSOS (escrito en Python, desarrollado por el Istituto Scienze della Terra) y la implementación desarrollada por 52North (en Java), ambas proyectos Open Source.

Gracias a los estándares para la gestión de sensores que forman SWE y el software creado por estos grupos de desarrollo, hoy en día existen proyectos como MMI (iniciativa para establecer observatorios oceánicos), GITEWS (un sistema para la detección y alerta de tsunamis) o S@NY (proyecto europeo centrado en redes de sensores in-situ para la monitorización del entorno). [5]

1.1.2. RBAC y modelos basados en él

Aunque desde la década de los 70 ya existían formas rudimentarias de control de acceso basado en roles, no fue estandarizado hasta el año 2004 cuando el Instituto Nacional de Estándares y Tecnología de Estados Unidos (NIST) adoptó la propuesta realizada por Sandhu, Ferraiolo y Kuhn, que empezaron a estudiar este modelo de control de acceso desde principios de los 90. A pesar de haber tardado tanto es ser estandarizado, muchos sistemas utilizan de una forma u otra, su propia implementación de RBAC.

Actualmente, RBAC es utilizado en diferentes ámbitos, por ejemplo, en sistemas de gestión de bases de datos relacionales (como Oracle y Sybase) o en sistemas de administración de seguridad en empresas (como Tivoli Identity Manager). También se están desarrollando servicios web RBAC, así como un esquema XML. [6]

RBAC es capaz de emular lo que ofrecen los modelos de control de acceso tradicional como MAC (Mandatory Access Control) o DAC (Discretionary Access Control) [7] , sin embargo, tiene sus limitaciones y no es capaz de ofrecer lo que otros sistemas de control de acceso como ABAC (Attribute Based Access Controls) o CapBAC (Capabilities Based Access Control), sobre todo en términos de escalabilidad. [8]

Una de las grandes ventajas que ofrece el modelo RBAC es que permite poder tomarlo como base para construir otros modelos de control de acceso. Se habla de extensiones del modelo RBAC cuando otros modelos de control de acceso contienen RBAC pero teniendo en cuenta otros factores a la hora de otorgar un permiso [6]. Actualmente existen múltiples modelos basados en RBAC, como por ejemplo: TRBAC (Temporal RBAC) [6], ERBAC (Enterprise RBAC) o TIE-RBAC (RBAC modificado para redes sociales). De entre las modificaciones existentes de RBAC existen aquellas que toman en consideración de alguna manera la posición del usuario, entre ellas cabe destacar PROX-RBAC (Proximity-based RBAC), STAR-BAC (SpatioTemporal RBAC) y GEO-RBAC (Geo-spatial RBAC) [9] [10] [11]. En la siguiente tabla comparativa podemos observar las diferencias entre algunos modelos de RBAC que toman en consideración la posición del usuario: [12]

Características	GRBAC	GSAM	GEO-RBAC	STRBAC	GSTRBAC	ESTARBAC	GeoXACML
Base formal	Sin formalizar	Teórico	Teórico	Teórico	Teórico y lógica de predicados	Teórico	Teoría de lógica descriptiva cuestionable
Soporte de control de acceso basado en roles	RBAC3	Sin RBAC. Jerarquías de tipo geoespacial y credencial	RBAC3	RBAC3	RBAC3	RBAC0	Perfil XACML RBAC
Modelo de datos espacial	Ninguno	Mapas de vectores y rasterizado digital de imágenes	Modelo geométrico referenciado	Espacio geométrico tridimensional	Ninguno	Ninguno	Modelo de acceso simple a características geométricas (OGC)
Granularidad espacial	Sin especificar	Múltiples resoluciones de imágenes	Punto, línea, polígonos y listas combinadas	Puntos físicos, localizaciones físicas y lógicas	Sin especificar	Extensión espacial y puntos físicos	Múltiples clases geométricas
Restricciones temporales	Sí	Limitado	No	Sí	Sí	Sí	Posible
Especificación de normas	Ninguno	Colecciones	Sin especificar	Colecciones	Permitido	XML	Lenguaje de normas GeoXACML
Administración de normas	No	No	Existen propuestas avanzadas que lo posibilitan	No	No	No	Sí
Integración de normas múltiple	No	No	No	No	No	No	Sí
Control de acceso físico	Posible	No	Posible	Posible	Sí	Posible	Posible
Control de acceso lógico	Posible	Sí	Sí	Sí	Sí	Sí	Posible

Figura 1.2: Tabla comparativa de algunos modelos espaciales de RBAC

1.2. Motivación

La interconexión a través de internet de todo tipo de objetos y entidades, conocido como Internet de las Cosas (IoT, por sus siglas en inglés) plantea nuevos retos para la seguridad, que requiere revisar y renovar las soluciones existentes o desarrollar nuevas. Entre las soluciones y mecanismos que necesitan ser revisados, se encuentran los sistemas de control de acceso. Entre algunos de estos retos destacamos los relacionados con la gestión, escalabilidad y la inclusión de nuevas propiedades a la hora de realizar el control de acceso. []

Una de las propiedades a tener en cuenta cuando hablamos de IoT es la localización física de los elementos que forman la red, según en qué entornos, tener conocimiento de la posición puede ser crítico para detectar y prevenir vulnerabilidades y fallos de seguridad.

Uno de los planteamientos posibles para poder gestionar esta gran cantidad de elementos es creando un sistema que utilice los modelos, servicios y estándares SWE desarrollados por OGC, ya que proveen de las herramientas necesarias para poder construir nuevos servicios y sistemas y que éstos sean interoperables entre sí. De esta forma, cada elemento se convierte en un sensor móvil, al que se le pueden ir añadiendo diferentes propiedades y atributos dependiendo del tipo de elemento.

Este proyecto intentará realizar una primera aproximación para solventar uno de los problemas de seguridad que plantea IoT, el control de acceso. Para ello, adaptaremos el modelo de control de acceso basado en roles para que pueda ser integrado en el sistema planteado; en concreto, utilizaremos la posición del sensor como pieza fundamental a la hora de realizar el control de acceso, por tanto, escogeremos GEO-RBAC como modelo a seguir.

1.3. Objetivos y requerimientos de la aplicación

El hilo conductor de este proyecto será, como ya se ha comentado, el diseño del procesador de eventos que forma el núcleo de un sistema GEO-RBAC, en el cual se procesarán la posición e identificación de los usuarios para generar las alertas correspondientes, quedándose fuera de los límites de este proyecto el diseño de las entradas (la aplicación que envía las posiciones en los elementos) y el de las salidas (la aplicación que recibe e interpreta las alertas).

Por tanto, podemos establecer como objetivos principales de este proyecto:

- **Desarrollar un procesador de eventos GEO-RBAC integrable en cualquier sistema:** El procesador de eventos ha de ser capaz de recibir entradas desde cualquier fuente y ha de proporcionar un API para que las alertas puedan ser accedidas por otras aplicaciones.
- **Integrar el procesador de eventos GEO-RBAC en un sistema de sensores:** Se ha de desarrollar e implementar la opción de que las entradas sean ofrecidas por un SOS.
- **Realizar pruebas de escalabilidad e interpretar los resultados:** De esta forma, podremos revisar las debilidades de la aplicación y poder establecer las necesidades que se deben cubrir en ciclos posteriores del desarrollo.

Una vez establecidos los objetivos principales, podemos deducir una serie de objetivos secundarios:

- **Familiarizarse con un procesador de eventos complejos**
- **Aprender cómo interaccionar con un SOS**
- **Aprender a desarrollar proyectos colaborativos**
- **Aprender a utilizar herramientas para realizar pruebas de estrés**

Los objetivos secundarios son valores añadidos que se obtendrán tras finalizar del proyecto. Además, para poder lograr los objetivos principales propuestos, la aplicación debería cumplir los siguientes requisitos:

- **Escalabilidad:** La aplicación ha de ser capaz de procesar una gran cantidad de eventos de forma simultánea con un tiempo de respuesta razonablemente bajo.
- **Flexibilidad:** El sistema debería ser capaz de adaptarse a las necesidades del entorno en el que se ejecutará.
- **Extensibilidad:** Posibilidad de ampliar su funcionalidad a través de interfaces y APIs.

Además de los requrimientos principales, también sería conveniente que la aplicación sea:

- **Óptima:** Dentro de lo posible, se intentará optimizar el código para que tenga menos coste computacional, aunque intentando que el código sea lo más claro posible.
- **Independiente:** Sería conveniente que la aplicación pueda ser ejecutada en cualquier sistema operativo, sin tener que recompilar el código cada vez.

Capítulo 2

Tecnologías, servicios y herramientas utilizadas

Con el fin de cumplir los requerimientos expuestos en el capítulo anterior, hemos escogido las siguientes tecnologías y herramientas para poder desarrollar la aplicación y realizar las pruebas de estrés.

2.1. Tecnologías

En esta sección se describirán las diferentes tecnologías (lenguajes de programación, sistemas de gestión de bases de datos, formatos de intercambio de información y protocolos de conexión) que se van a utilizar para el desarrollo del proyecto.

2.1.1. Java

Java es un lenguaje de programación que nació en 1991 por la empresa Sun Microsystems (adquirido por Oracle en 2010). Su propósito principal fue crear un lenguaje que cumpliese con la filosofía conocida, por sus siglas en inglés, como WORA ("Write Once, Run Anywhere"), que implica que el código pueda ser ejecutado por cualquier dispositivo sin necesidad de tener que recompilar o adaptar el código.

Esto es posible porque el código es pseudocompilado, es decir, se traduce a Java bytecodes que puede ser interpretado por cualquier máquina virtual de Java (JVM). Por contra, pierde la utilidad de poder utilizar recursos de bajo nivel, que otros lenguajes como C o C++ sí permiten.

Entre sus numerosas ventajas se pueden destacar [13]:

- **Independiente de la plataforma:** Como ya hemos comentado, su código es ejecutado por la máquina virtual de Java lo que permite que no tenga que ser recompilado para cada plataforma.
- **Orientado a objetos:** Nos permite crear programas modulares y reusar el código escrito.
- **Distribuido:** Java fue diseñado con capacidad de comunicarse por red de forma inherente, de forma que se pueden crear aplicaciones distribuidas y compartir recursos de forma sencilla.
- **Robusto:** A la hora de compilar el código a Java bytecodes, el compilador intenta asegurar al máximo que pueda ser ejecutado sin problemas, detectando los posibles errores que puedan ocurrir en tiempo de ejecución.
- **Multihilo:** Java tiene integrada la habilidad de ejecutar código concurrente, y realizar múltiples tareas de forma simultánea.
- **Dinámico y extensible:** En Java, cada clase está, normalmente, contenida en un único fichero, lo que permite cargarlas en tiempo de ejecución de forma dinámica. Ésto permite crear programas modulares con la habilidad de ser extendidas.
- **Eficiente:** Aunque el código no sea compilado directamente a instrucciones del procesador, Java permite, en la mayoría de los casos ejecutarse con la suficiente eficiencia para minimizar el impacto. Además, gracias al "recolector de basura" evita por completo las fugas de memoria, ya que elimina los objetos que dejan de ser referenciados.

Java se distribuye en diferentes formatos, cada uno orientado a diferentes propósitos y así poder dar mayor soporte [13]:

- **Java SE:** Java Standard Edition, es la distribución de propósito general, está orientado al usuario final sin ofrecer recursos enfocados a usos específicos.
- **Java EE:** Java Enterprise Edition está orientado para diseñar software para empresas, ofrece soporte para crear aplicaciones en red y servicios web escalables, fiables y seguras.

- **Java ME:** Java Micro Edition es utilizado para integrar Java en sistemas embebidos que sean poco potentes. Se enfoca en consumir los menos recursos posibles. Fue utilizado sobre todo por teléfonos móviles, pero hoy en día está cayendo en el desuso debido a sus limitaciones.

- **JavaFX:** JavaFX actualmente se encuentra integrado en la octava versión de JavaSE, pero hasta entonces ha sido distribuida de forma separada. Ofrece la posibilidad de crear interfaces gráficas mucho más completas que las que ofrece por defecto JavaSE. Utiliza un esquema modelo-vista-controlador y también tiene incorporado la posibilidad de crear aplicaciones web gráficas.

Además, el entorno de ejecución de Java se distribuye de forma separada al entorno de programación, de forma que cada uno de los formatos vistos es distribuido mediante un JDK (por sus siglas en inglés, Java Develpment Toolkit) y puede ser ejecutado por un JRE (por sus siglas en inglés, Java Runtime Environment) y así el usuario final sólo debe instalar el JRE para poder ejecutar una aplicación Java desarrollada en cualquiera de sus formatos.

Para este proyecto, se ha decidido utilizar JavaEE para desarrollar la aplicación, ya que, como veremos más adelante, necesitaremos hacer uso de algunas de sus librerías.

2.1.2. SQLite

SQLite es un sistema de gestión de bases de datos relacional (RDBMS) de dominion público que se lanzó a mediados del año 2000. Su principal diferencia con la mayoría de sistema de gestión de bases de datos es su pequeño tamaño y que la base de datos entera se guarda en un sólo fichero que, en su última versión estable hasta la fecha (versión 3) puede ser de hasta 2 Terabytes. Este diseño es posible porque el fichero que contiene la base de datos se bloquea al principio de cada transacción, asegurando la integridad de los datos.

SQLite es compatible con ACID, un acrónimo inglés de Atomicity (Atomicidad), Consistency (Consistencia), Isolation (Aislamiento) and Durability (Durabilidad), esto implica que es capaz de hacer transacciones y permite hacer “rollback” (deshacer los cambios) en caso de conflicto o acceso concurrente. Aunque originalmente está escrito en C, al ser de dominio público existen bibliotecas para prácticamente todos los lenguajes de programación que permiten utilizar bases de datos SQLite.

Debido a su pequeño tamaño, SQLite es especialmente adecuado para sistemas integrados y se incluyen en la mayoría de sistemas operativos móviles y navegadores web. Para este proyecto, se ha escogido SQLite por su simpleza, pues no se van a utilizar demasiadas tablas. [14]

2.1.3. XML y WSDL

XML viene de sus siglas en inglés eXtensible Markup Language (lenguaje de marcas flexible), es un leguaje de marcado desarrollado por el World Wide Web Consortium (W3C). El origen de XML se remonta a los años setenta, cuando IBM desarrolló un lenguaje conocido como GML (Generalized Markup Language) para poder almacenar información, este lenguaje se normalizó por la ISO en 1986, surgiendo SGML (Standard GML) con mayor capacidad de adaptación. Cuando se creó la web en 1989 se creó también el lenguaje HTML, definido con el estándar SGML.

El uso extensivo de HTML resaltó los problemas que tiene SGML: cada documento pertenece a un vocabulario fijo y cada navegador web puede interpretarlo de forma libre, incluso aunque el documento contenga errores. XML se creó como un subconjunto de SGML con la intención de permitir mezclar diferentes lenguajes (es decir, que sean extensibles) y centrándose en que el documento no tenga errores de sintaxis para que los analizadores pudiesen ser más simples, esta sintaxis viene definida en un esquema XML que describe la estructura y restricciones del formato.

Gracias a su flexibilidad, XML puede ser utilizado para el intercambio de información estructurada entre diferentes plataformas, y permitir la compatibilidad entre sistemas distintos. Entre otros usos, cabe destacar el formato WSDL. WSDL describe, en un formato basado en XML, la interfaz pública de los servicio Web: requisitos del protocolo, formato de los mensajes y operaciones soportadas. [15] [16] [17] [18]

En este proyecto se utilizará WSDL para intercambiar información con el SOS. Ya que todos los modelos desarrollados por OGC están descritos como esquemas XML y las interfaces públicas de los servicios web como el SOS están definidos mediante documentos WSDL.

2.1.4. JSON

Como XML, JSON es un formato ligero para el intercambio de datos. Es un acrónimo de JavaScript Object Notation y fue creado para poder intercambiar

y almacenar objetos de Javascript. Actualmente, se utiliza como alternativa a XML para el intercambio de datos (no sólo objetos Javascript) ya que su sintaxis es mucho más ligera que la utilizada por XML.

JSON es mayormente utilizado para el intercambio de datos dinámicos dentro de las páginas web a través de Javascript, pues la conversión de un objeto Javascript a JSON y viceversa es automática y, por tanto, mucho más sencilla a la hora de implementar que utilizar XML para intercambiar los datos. Además, muchos servicios web soportan también JSON para realizar este intercambio de datos, y JSON tiene soporta en prácticamente todos los lenguajes de programación a través de librerías. [19]

Se hará uso de JSON en este proyecto para enviar los datos generados a los puntos de salida, de forma que se codificará la información que se quiere enviar en este formato y se enviarán como una cadena de texto.

2.1.5. WebSocket

WebSocket es una tecnología relativamente joven que está siendo estandarizada por el W3C (el API) y por el IETF (el protocolo). Proporciona una comunicación bidireccional, full-duplex sobre un único socket TCP. Aunque su diseño está enfocado para ser implementado en navegadores y servidores web, realmente se puede utilizar por cualquier aplicación cliente/servidor. Actualmente ya se encuentra implementado en las últimas versiones de los navegadores más usados, y su API forma parte del conjunto de APIs de HTML5.

Su principal ventaja se debe a que permite a las páginas web establecer este tipo de conexiones TCP bidireccionales y persistentes, ya que hasta ahora sólo era posible simularlo con tecnologías como Push o Comet, pero estas poseen tres principales desventajas en comparación: la primera es que no son están estandarizados (dependen de librerías desarrolladas por terceros), la segunda es que son más complicados de implementar y la última y más importante es que la persistencia de la conexión es una simulación, el truco consiste en realizar lo que se conoce como Long Polling (sondeo largo), se abre una conexión HTTP desde el cliente al servidor y no se cierra hasta que se reciban datos nuevos. Esta método para crear la ilusión de una conexión persistente conlleva una sobrecarga de HTTP, por lo que las aplicaciones que requieren baja latencia serían incapaces de funcionar correctamente, y eso sin contar con la penalización del “slow-start” cada vez que se abre una conexión TCP.

Para realizar una conexión WebSocket, el cliente manda una petición HTTP

GET indicando como protocolo ws:// (o wss:// para la versión segura) añadiendo los campos "Connection: Upgrade", "Upgrade: WebSocket" y las cabeceras necesarias para realizar el intercambio de tokens del handshake. Si el servidor responde aceptando la conexión ésta ya no se cierra, y el intercambio de mensajes será a partir de ahora la definida por el protocolo WebSocket. [20]

En este proyecto se utilizarán conexiones WebSocket para establecer comunicación con los clientes que quieran recibir los datos en tiempo real, y de esta forma se simplifica la posibilidad de crear un cliente web mientras que si se decidiese crear un cliente en cualquier otra tecnología tan sólo se ha de utilizar una librería que dé soporte para establecer conexiones WebSocket.

2.1.6. JavaScript

Es un lenguaje de programación interpretado orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Fue desarrollado originalmente por un desarrollador de Netscape con el nombre de Mocha, más tarde se renombró a LiveScript y finalmente quedó como Javascript, coincidiendo con el momento en que Netscape empezó a dar soporte a Java en su navegador. Esto creó bastante confusión pues parece que se trata de un lenguaje derivado de Java, pero no están relacionados ni en semántica ni en propósito (solo adopta algunos nombre y convenciones).

Actualmente se utiliza principalmente en los clientes web, permitiendo crear interfaces de usuarios más ricas y páginas web dinámicas interactuando con con una implementación del Document Object Model (DOM) y mediante el uso de APIs que dan soporte a la utilización del sistema de archivos, manejar la reproducción de archivos multimedia, crear gráficos complejos, WebSockets, etc... Una de las funcionalidades más utilizadas es su soporte para realizar peticiones HTTP de forma asíncrona (conocido como AJAX). [21]

Javascript utiliza de forma nativa JSON para el intercambio de mensajes y objetos, es por ello que lo utilizaremos en este proyecto para desarrollar herramientas para verificar el correcto funcionamiento de la aplicación ya que simulan los dispositivos de entrada y el cliente de salida, pues al usar WebSockets y mensajes en formato JSON es especialmente útil.

2.1.7. Scala

Scala es un lenguaje de programación multiparadigma: funcional, orientado a objetos, imperativo y concurrente. Su código fuente se compila a Java byte-

codes, y sus ejecutables se ejecutan en la máquina virtual de Java, pudiendo utilizar sus librerías (y viceversa) ya que es compatible con las aplicaciones de Java existentes.

Scala es relativamente joven, su diseño empezó en 2001 y se publicó en 2004, originalmente también daba soporte a la plataforma .NET aunque se en 2012 lo dejó. Aunque su sintaxis es parecida a la de Java, ofrece, entre otras mejoras, una mayor flexibilidad, soporte sintáctico para programación funcional y mayor simpleza en algunas expresiones. [22]

Que Scala pueda ser compatible con Java ofrece la posibilidad de poder combinar ambos lenguajes para poder crear aplicaciones extensibles, por ejemplo, se puede crear una aplicación en Java que permita cargar código en Scala y ejecutarlo. Será necesario utilizar Scala en este proyecto para poder programar la aplicación que utilizaremos para realizar pruebas de estrés y que describiremos en la sección de herramientas.

2.2. Servicios

Como se explicará en los siguientes apartados, la aplicación que se va a desarrollar hace uso de dos servicios distintos: uno utilizado para recibir datos, el otro para enviarlos. Vamos describirlos brevemente a continuación.

2.2.1. SOS

Como ya se ha dicho en el estado del arte, el servicio web SOS es un estándar de la OGC que define la interfaz web para realizar lo que se conoce como CRUD (Create, Read, Update, Delete) a los sensores y los datos obtenidos por ellos.

Para modelar los sensores se utiliza la especificación SensorML 2.0, y para modelar los datos de los sensores se utiliza O&M 2.0 (Observations and Measurements), son esquemas XML que nos describen la sintaxis que se ha de seguir para poder modelarlo correctamente.

Para poder describir las operaciones que permite realizar el SOS, primero se ha de entender con qué metadatos se pretende describir los sensores y la información medida. En la siguiente imagen se pueden observar los metadatos necesarios para insertar una medida: [23]

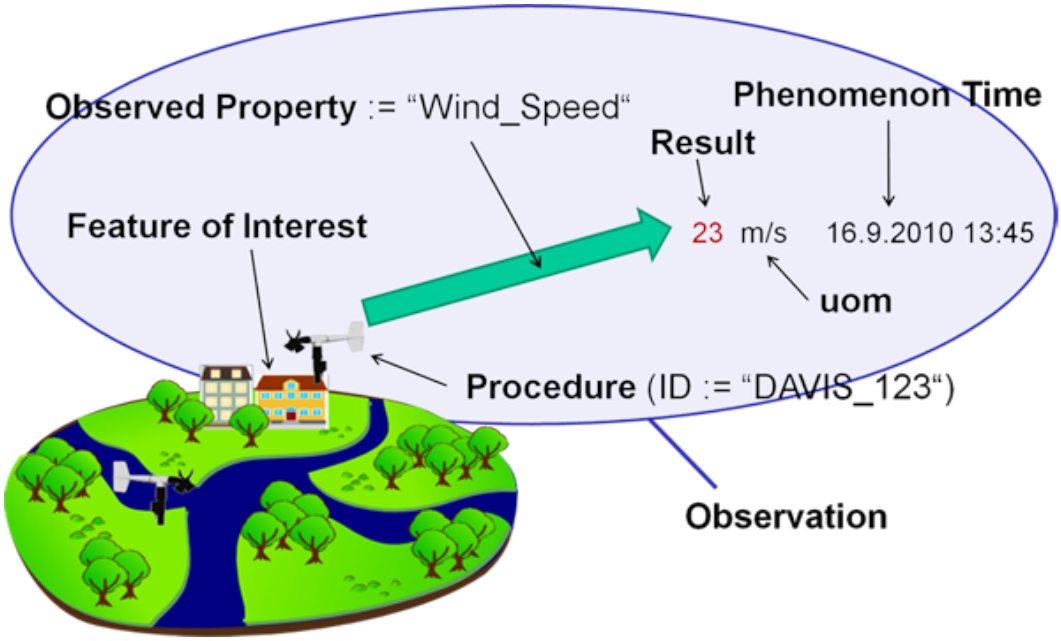


Figura 2.1: Metadatos de los sensores y medidas [23]

- **Feature of interest:** Es el "objeto" de las medidas, normalmente perteneciente al mundo real, como puede ser la presión atmosférica en una cierta región o el instrumento que lo mide.
- **Phenomenon time:** Es el instante temporal en el que se considera realizada la medida.
- **Procedure:** Es el proceso que realiza la observación, normalmente suele ser un sensor real (por ejemplo, un termómetro) pero también puede ser virtual (por ejemplo, un proceso computacional que realiza simulaciones).
- **Result time:** Es el instante temporal en el que se ha computado el resultado de la medida. Esta información es relevante cuando la medida se computa como una combinación de otras, ya sean del mismo tipo (medias, varianzas, etc...) o de distinto tipo (medidas indirectas, como por ejemplo, tensión a partir de corriente y carga).
- **Observed property:** Es la descripción descripción de la propiedad observada (por ejemplo, "presión atmosférica", "velocidad del viento").
- **Result:** Es el valor del resultado de la observación (puede ser escalar o un vector multidimensional).

- **uom:** Es la unidad de medida, para poder interpretar el valor del resultado de forma correcta. Se suelen utilizar valores UCUM (Unified Code for Units of Measure, Código Unificado para Unidades de Medida).

Las operaciones que debe soportar una implementación del estándar SOS 2.0 deben ser las siguientes: [24]

- **GetCapabilities:** Responde con una descripción completa del servicio web, (funcionalidades, contacto, etc...)
- **GetObservation:** Devuelve datos provenientes de los sensores codificados según el esquema O&M 2.0
- **DescribeSensor:** Esta petición devuelve la descripción del sensor pedido en según el esquema SensorML.
- **GetFeatureOfInterest:** Obtiene la descripción del objeto o propiedad que se está midiendo.
- **GetObservationById:** Obtiene una única observación determinada por su ID.
- **InsertResultTemplate:** Se puede insertar una plantilla para que las sucesivas inserciones de resultados sean más sencillas.
- **InsertResult:** Esta petición permite insertar resultados de medidas en el SOS, si hay una plantilla insertada esta petición se simplifica bastante.
- **GetResultTemplate:** Se utiliza para obtener la estructura con la que se devuelven los resultados.
- **GetResult:** Con esta petición se pueden obtener los resultados que han sido insertados en el SOS. Si previamente se ha obtenido la estructura de los resultados, la respuesta se simplifica bastante.
- **InsertSensor:** Permite añadir nuevos sensores.
- **InsertObservation:** Permite añadir nuevas observaciones.
- **UpdateSensorDescription:** Con esta petición se puede actualizar la descripción de un sensor que ya haya sido insertado.
- **DeleteSensor:** Con esta petición se pueden los datos relacionados con un sensor.

En este proyecto utilizaremos las peticiones GetCapabilities y GetObservation, la primera para obtener un listado de los sensores que se encuentran registrados en el SOS, y la segunda para obtener la última posición enviada por el sensor. Aunque también se hará uso de las peticiones que permiten registrar sensores (InsertSensor) e insertar medidas (InsertObservation) para realizar pruebas y verificar el funcionamiento.

2.2.2. GCM

Google Cloud Messaging es un servicio que permite enviar datos desde servidores gestionados por los desarrolladores de una aplicación a dispositivos móviles con el sistema operativo Android (versión 2.2 o superior) o a navegadores web Google Chrome.

La funcionalidad principal que ofrece GCM es que los mensajes son almacenados y gestionados por ellos, de forma que aunque los dispositivos a los que se quiere enviar un mensaje no están disponibles (no tienen acceso a la red de datos), GCM encolará los mensajes y la próxima vez que estén disponibles serán notificados de la existencia de estos mensajes, de forma que los pueden recuperar. Además, si la aplicación que tenga mensajes por recibir en ese momento no está abierta (o está "dormida") Android iniciará la aplicación y le entregará los mensajes. [25]

El servicio permite recibir peticiones HTTP POST por parte de los servidores que gestionan la aplicación destino con los siguientes parámetros (en formato JSON):

- **registration_ids:** En este parámetro se especifica un lista de identificadores de los dispositivos android que recibirán el mensaje.
- **notification_key:** Este parámetro contiene los identificadores de los usuarios, donde cada usuario puede estar asociado a múltiples dispositivos.
- **collapse_key:** Si se quieren enviar múltiples mensajes con la misma finalidad, con este parámetro se pueden categorizar los mensajes de forma que si el dispositivo no está disponible, la próxima vez que se conecte recibirá solamente el último mensaje de cada categoría. Y así evitar recibir mensajes que podrían carecer de sentido.
- **data:** Los datos a ser enviados, han de ser un objeto JSON que con parejas de clave: valor. El valor ha de ser, preferiblemente, una cadena de caracteres.

- **delay_while_idle:** En este parámetro se indicará si se desea enviar el mensaje en cuanto el dispositivo esté activo en caso de no estarlo.
- **time_to_live:** El tiempo (en segundos) que se debe guardar el mensaje en caso de no estar activo el dispositivo (el máximo son cuatro semanas).
- **restricted_package_name:** Con este parámetro se puede configurar que el mensaje sea enviado solamente a los dispositivos indicados en registration_ids que además tenga la aplicación instalada.
- **dry_run:** Este parámetro sirve para indicar que el mensaje es de prueba y que no sea retransmitido.

Además, en la cabecera de la petición HTTP POST se ha de añadir el campo Authorization cuyo valor será la clave de desarrollador que ofrece Google para este servicio. GCM también permite la posibilidad de enviar los mensajes como texto plano en lugar de en formato JSON, aunque no está recomendado. También da soporte para una comunicación bidireccional (entre la aplicación y los servidores de la aplicación) a través de los servidores de GCM utilizando el protocolo CCS (derivado de XMPP). [26]

Para este proyecto se utilizará GCM para poder enviar mensajes a los dispositivos, se detallará en los capítulos posteriores.

2.3. Herramientas

Para poder llevar a cabo el desarrollo, ejecución prueba de la aplicación, se ha hecho uso de diferentes herramientas, se describirán las más importantes a continuación.

2.3.1. Eclipse

Eclipse es un entorno de programación integrado pensado para desarrollar aplicaciones en Java. Es completamente modular y puede ser extendido mediante plugins, de esa forma es capaz de dar soporte para programar en prácticamente cualquier otro lenguaje: Ada, C, C++, Fortran, Haskell, JavaScript, PHP, etc...

El proyecto Eclipse es desarrollado por Eclipse Foundation, creado en 2004 por más de 80 miembros (entre ellos, IBM, Red Hat y SuSE) y que distribuye

el software en código abierto. La primera versión distribuida fue la 3.0 y cada año desarrollan una nueva versión que integra de forma nativa el último JDK estable de Java. [27]

Gracias a su arquitectura totalmente modular existen plugins capaces de dar soporte para frameworks (Spring, AppEngine, Ruby On Rails...), aplicaciones de red (clientes Telnet, SSH, FTP, SCP, etc...), herramientas de colaboración (CVS, SVN, GIT, Mercury, Mylyn,...) y prácticamente cualquier tecnología relacionada con el desarrollo de aplicaciones.

El área de trabajo de Eclipse es completamente configurable y está dividido en diferentes "vistas". A continuación se explicará la configuración de las "vistas" del área de trabajo utilizada para desarrollar la aplicación de este proyecto:

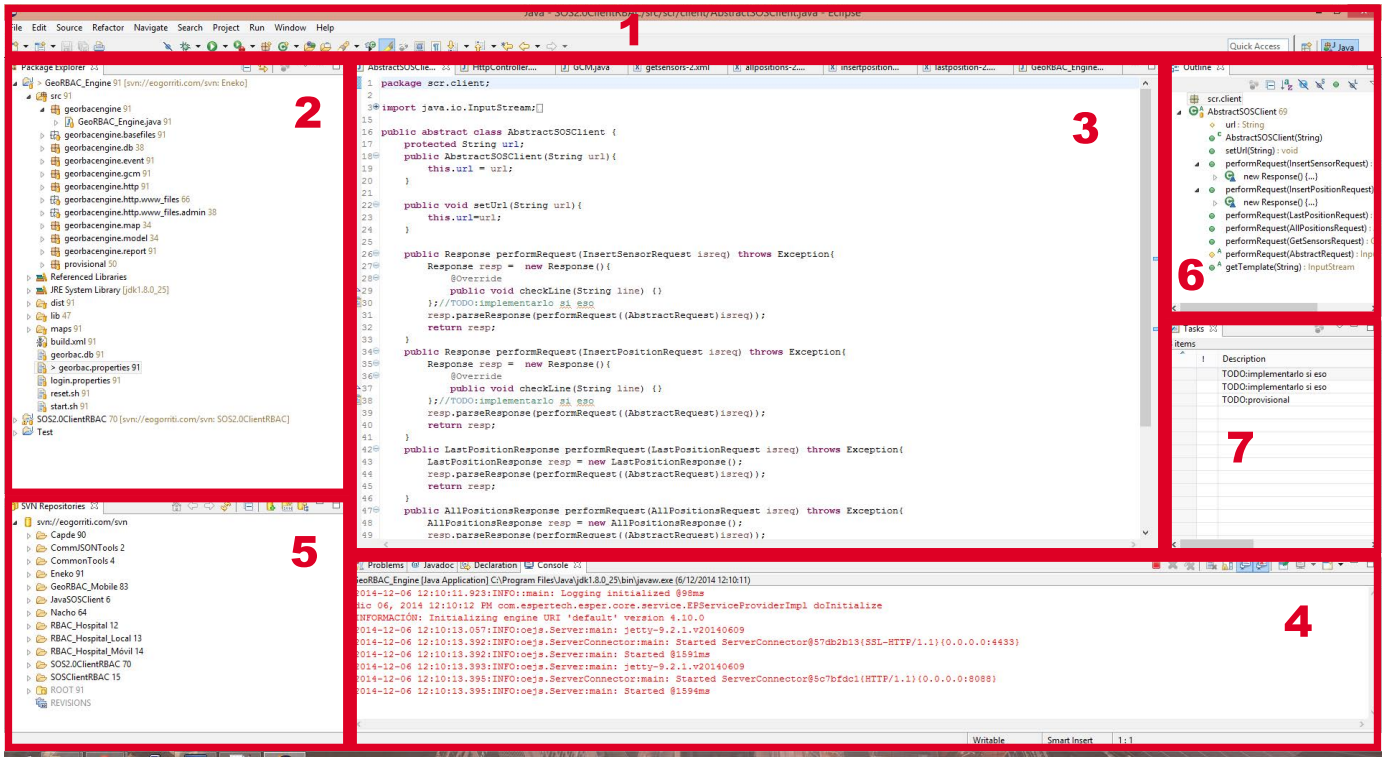


Figura 2.2: Configuración de las vistas de Eclipse

- **1:** Es la barra de herramientas, su posición es fija pero las herramientas de la barra es configurable, entre ellas destacan las herramientas permiten

compilar/ejecutar/debugear el proyecto y los botones para crear nuevas clases/paquetes.

- **2:** Es el explorador del proyecto, donde se muestra la estructura de ficheros del proyecto y las librerías referenciadas. Dentro de la carpeta del código fuente (carpeta "src") se muestran los paquetes y clases que forman la aplicación.
- **3:** Ésta es la vista principal del área de trabajo, donde se puede editar el código fuente del archivo seleccionado. Mediante el uso de pestañas se facilita el cambio de un archivo a otro.
- **4:** Normalmente se sitúa en este lugar la consola de salida, donde se puede visualizar la salida estándar y la salida de error de la aplicación que se esté ejecutando.
- **5:** En esta vista se ha situado el explorador SVN que permite gestionar el repositorio donde se almacena el proyecto.
- **6:** La vista "Outline" provee una visualización global de la clase que representa el archivo abierto, de forma que es más sencillo navegar a los diferentes puntos importantes de la clase de forma rápida. Es especialmente útil cuando la clase contiene muchas líneas de código.
- **7:** En esta vista se sitúa la lista de tareas activas para el archivo seleccionado, para así poder navegar rápidamente a los puntos concretos donde falta completar o modificar el código.

2.3.2. Maven

Maven es una herramienta utilizada para automatizar el proceso de compilación de las aplicaciones escritas en Java (también soporta otros lenguajes). También permite gestionar el control de dependencias y a través de repositorios centrales, como Maven 2 Central Repository, es capaz de descargar automáticamente las librerías necesarias para la compilación.

A través de un fichero XML, conocido como Project Object Model (pom.xml), se puede configurar los datos de la aplicación, las librerías de las que depende y también el proceso de compilación. En Maven, el proceso de compilación no solamente compila la aplicación, sino que es capaz de realizar muchas más cosas. En el archivo de configuración se ha de configurar lo que se conoce como el ciclo de vida de construcción, donde se pueden configurar distintos procesos previos y posteriores a la compilación del código de fuente: validación, generar

documentación, realizar un test de funcionamiento (mediante una herramienta de testeo, como JUnit), empaquetar la aplicación con o sin dependencias, etc... [28]

A la hora de desarrollar la aplicación descrito en este proyecto, se comenzó utilizando la herramienta Apache Ant para automatizar el proceso de compilación, pero posteriormente se utilizó Maven, ya que se integra mejor con Eclipse mediante un plugin, y es capaz de resolver las dependencias.

2.3.3. SVN

Subversion (SVN) es una herramienta de control de versiones de código abierto. Fue desarrollado por la fundación Apache con el objetivo de suceder a CVS, y extender sus funcionalidades (pero manteniendo la compatibilidad).

SVN es un sistema centralizado, lo que significa que el los proyectos son subidos a un repositorio central y realizar en el repositorio las operaciones necesarias, a diferencia de las herramientas de control de versiones distribuidas (como Git) donde los repositorios están distribuidos.

Las operaciones en SVN son atómicas, esto quiere decir que si fallan o son interrumpidas los datos no se corrompen, sino que vuelven al estado original. Existen tres formas distintas de acceder al repositorio: de forma local, de forma remota mediante WebDAV (mediante http:// o https://) o mediante su protocolo propio (svn:// o svn+ssh// para el transporte encriptado). [29]

En este proyecto se ha utilizado SVN para el control de versiones por su simpleza e integración completa como plugin de Eclipse.

2.3.4. Jetty

Jetty es una servidor HTTP (y contenedor de Servlets) escrito enteramente en Java, y es de código abierto. El proyecto comenzó en 1995 y en 2009 fue integrado dentro de la fundación Eclipse. Actualmente soporta los protocolos HTTP/1.1, WebSocket y SPDY. También soporta diferentes tecnologías de Java como JSP, JMX, JNDI u OSGi.

Jetty es utilizado como servidor Web en numerosos productos como Google App Engine, FUSE o Apache Spark. También es utilizado en numerosos frameworks, como por ejemplo, Hadoop. [30] [31]

Jetty puede ser implementado de dos formas distintas, como un contenedor de Servlets (como Apache Tomcat, en el que los Servlets se despliegan en un directorio y son servidos) o de forma integrada en una aplicación (como librería). En este proyecto se utilizará Jetty de las dos formas, ya que será necesario integrarlo en la aplicación para poder comunicarse con los clientes conectados, pero también se instalará como contenedor de Servlets para poder desplegar el SOS (el proceso será explicará en el capítulo correspondiente).

2.3.5. 52North SOS

De las diferentes implementaciones del Sensor Observation Service desarrollado por el grupo SWE de OGC, el más utilizado es el creado por 52North. Está escrito en Java y es de código abierto. Se distribuye como una una colección de Servlets, empaquetado en un archivo WAR (Web Application aRchive).

Originalmente daba soporte solamente para PostgreSQL, pero la última versión da soporte para diversos sistemas de gestión de bases de datos relacionales como Oracle, MySQL y Microsoft SQL Server. Además, ofrece diversos conectores para realizar las operaciones del SOS: KVP, SOAP, POX y JSON.

Soporta todas las operaciones de la especificación 2.0 de SOS, listadas en la sección de servicios, y además añade dos operaciones para facilitar un poco más el acceso: [32]

- **GetDataAvailability:** Responde con la disponibilidad de los datos para ciertas configuraciones.
- **DeleteObservation:** Borra una única observación determinada por una ID.

En este proyecto utilizaremos 52North como implementación del SOS, entre otros motivos, porque es más sencillo de desplegar (se desplegará sobre el contenedor de Servlets Jetty), porque da soporte a MySQL (será el sistema de gestión de bases de datos que utilizaremos para el SOS) y por su bajo consumo de almacenamiento (1.36 millones de observaciones ocuparán solamente 750MB), ya que cuando realizemos las pruebas de carga necesitaremos introducir numerosas observaciones.

2.3.6. Gatling

Gatling es una herramienta para realizar pruebas de carga y realizar análisis y medidas del rendimiento de varios servicios, aunque se centra en servicios

web. Está diseñado para que se uso sea sencillo y tenga buen rendimiento. Soporta el protocolo HTTP de forma nativa, aunque mediante extensiones puede soportar más protocolos.

Gatling se configura mediante "escenarios" escritos en Scala, en cada escenario se pueden configurar prácticamente todos los parámetros necesarios para realizar pruebas de carga: número de conexiones simultáneas cada segundo, contenido de las peticiones, perfiles distintos de carga, etc... También se pueden utilizar plantillas para que se simplifique la configuración.

Una vez realizada la prueba de carga, por defecto Gatling genera los resultados en documentos html, donde se pueden ver diferentes gráficas que muestran diferentes aspectos relacionados con la prueba: latencia de cada conexión, número de conexiones aceptadas y rechazadas, etc... [33]

Se utilizará Gatling para hacer pruebas de carga de la aplicación que vamos a desarrollar, pues es necesario hacer una estimación del número de usuarios activos que será capaz de soportar la aplicación.

Capítulo 3

Arquitectura

En este capítulo vamos a explicar el funcionamiento de la aplicación que se va a desarrollar. Primero se explicará el escenario para el cual se aplicará el sistema GEO-RBAC diseñado, y se mostrará el esquema del sistema y el esquema del caso de uso. Seguidamente se mostrará la arquitectura de la aplicación que se va a desarrollar, dividido por los bloques funcionales que la forman.

3.1. Escenario

Antes de que el sistema entre en funcionamiento, el administrador insertará los roles que podrán tomar los dispositivos. Seguidamente, configurará el mapa donde se desenvolverá el sistema, estableciendo las zonas donde se restringirá o se autorizará el acceso a cada rol. Finalmente, se registrarán los dispositivos asociándolo a un sensor en el SOS, y a la vez se introducirán en la base de datos, asociándolo a un rol de los que se hayan introducido.

Los dispositivos serán los sensores del SOS, y harán el papel de usuario en el sistema GEO-RBAC. El dispositivo enviará, de forma periódica, la posición al SOS que lo almacenará en su base de datos particular.

El procesador de eventos tomará los datos de entrada realizando peticiones periódicas al SOS, recibiendo un listado de los sensores registrados y la última posición enviada. También consultará en la base de datos el rol que toma cada sensor. Con estos datos de entrada generará los eventos de entrada, y los procesará (contrastándolo con las zonas configuradas) para generar los eventos de salida. Los eventos de salida serán generados únicamente si el dispositivo ha accedido a una zona no autorizada.

Los eventos de salida serán recibidos por el administrador, a través del interfaz gráfico de gestión (el administrador debería ser el único con acceso). También enviará los eventos de salida a los servidores de Google Cloud Service, para que se encargue de retransmitirlo a los dispositivos, para comunicar que se ha accedido a una zona restringida.

3.2. Esquema del sistema

Vamos a mostrar el esquema simplificado del escenario explicado en la sección anterior:

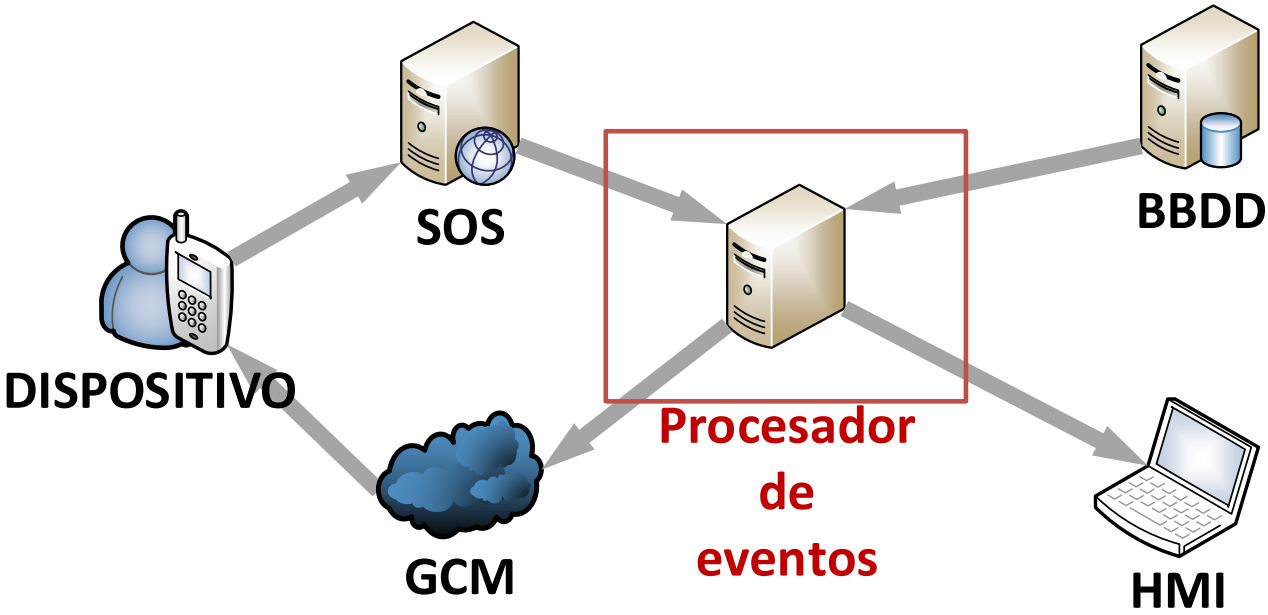


Figura 3.1: Esquema del sistema

Podemos observar los siguientes elementos:

- **Dispositivo:**

  Cualquier dispositivo capaz de establecer una conexión HTTP y enviar su posición. Si es un dispositivo Android podrá recibir los eventos retransmitidos por GCM.

- **SOS:**

  Un servidor HTTP donde esté desplegado un SOS y configurado para insertar, eliminar y actualizar sensores. También debe ofrecer la posibilidad de consultar el último estado actualizado de los sensores. Cada dispositivo representará un sensor con un identificador único.

- **BBDD:**

  Base de datos relacional con soporte para realizar consultas SQL. Deberá estar configurado con las tablas necesarias para asociar los sensores a los roles que pertenezcan.

- **Procesador de eventos:**

  Es la aplicación que queremos desarrollar. Recibirá como entradas la posición de los sensores y las tablas que relacionan los sensores con cada dispositivo y sus roles, genera un nuevo evento y lo procesará el CEP (Complex Event Processor) que distinguirá si el dispositivo se encuentra en una zona autorizada o no, en cuyo caso informará a los elementos conectados a sus salidas.

- **HMI:**

  Una aplicación con una interfaz gráfica de gestión que tenga soporte para establecer una conexión vía WebSocket para poder recibir los eventos en tiempo real. Los administradores del sistema deberían ser los únicos que

pueden acceder a la aplicación de gestión.

- **GCM:**

  El servicio que ofrece Google que utilizaremos para retransmitir los eventos de salida.

Cabe destacar que siguiendo el esquema propuesto aislamos los dispositivos del procesador de eventos, lo que aumentará la complejidad del sistema. Pero son varias las razones por las que se ha decidido realizar esta separación.

Utilizando el SOS como elemento intermedio para el envío de posiciones nos beneficia, por un lado, porque el procesador es capaz de controlar la cantidad de posiciones que recibirá del SOS y por tanto, ser capaz de responder mejor a una gran cantidad de dispositivos enviando sus posiciones de forma simultánea. Por otro lado, si la conexión entre el procesador de eventos y el SOS no puede ser comprometida, estamos añadiendo una capa más de seguridad. Además, en caso de que el procesador de eventos se caiga, al menos el SOS seguirá recibiendo y almacenando las posiciones de los dispositivos.

Para el caso de los dispositivos Android, utilizar GCM como elemento intermedio nos ofrecerá, de nuevo, una capa más de seguridad pero también la posibilidad de "despertar" la aplicación de Android cuando se reciba un nuevo evento, y en caso de que el dispositivo pierda la conexión a Internet, enviar los eventos la próxima vez que se conecte.

3.3. Esquema del caso de uso

Podemos extender el esquema del sistema para obtener el esquema del caso de uso:

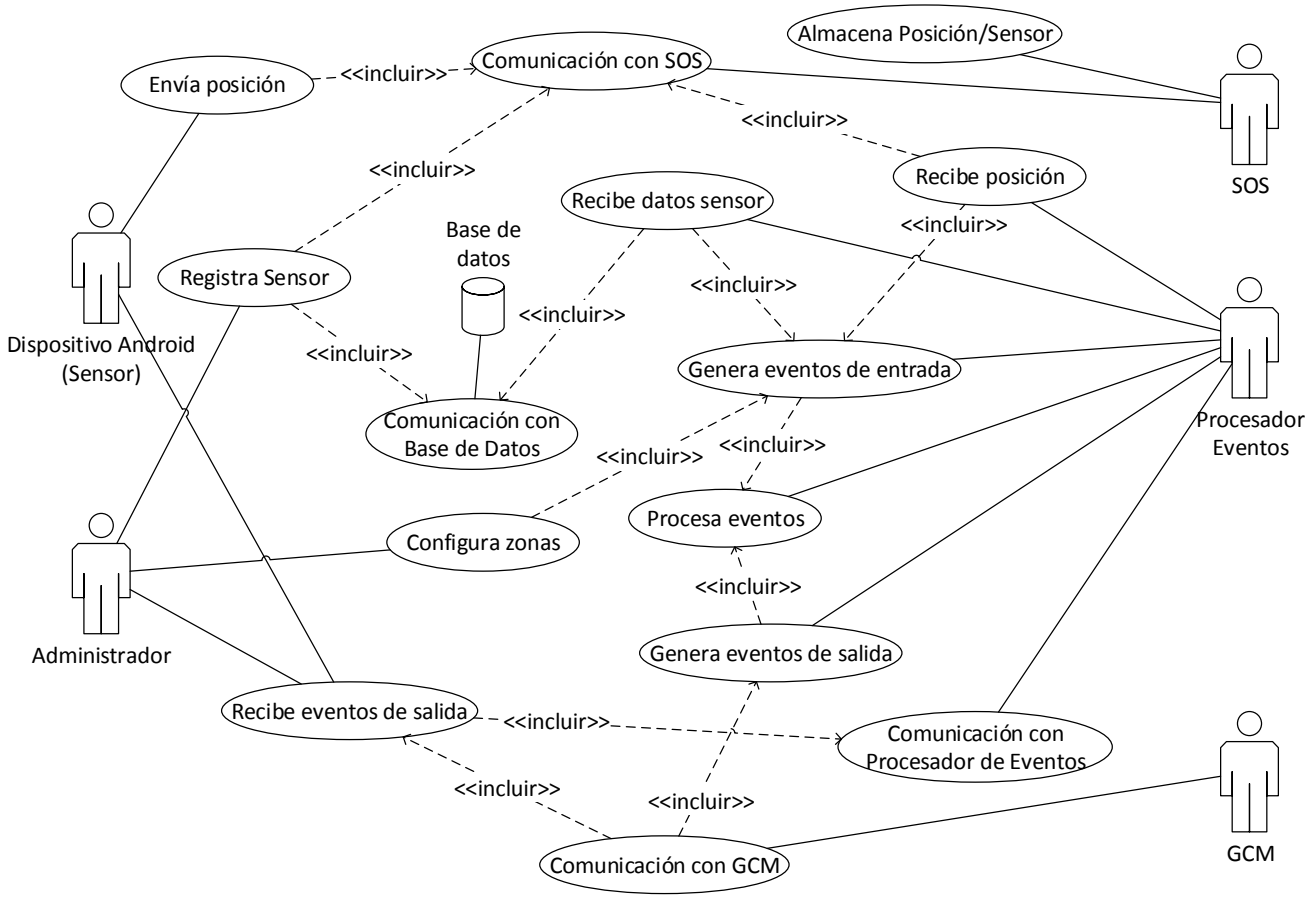


Figura 3.2: Esquema del caso de uso

Los actores del sistema son los mismos explicados en el esquema del sistema, pero el HMI se ha sustituido por el Administrador, ya que como se puede observar, realiza más acciones además de recibir los eventos de salida.

Según se puede ver en el esquema, los actores de la izquierda tienen una representación real: el administrador es la persona física encargada de gestionar el sistema, mientras que el sensor representa a la persona/objeto que será el usuario dentro del sistema. Los actores de la derecha y la base de datos son

servicios dentro del sistema y cada uno posee una interfaz de comunicación M2M (Machine To Machine).

3.4. Esquema del procesador de eventos

El siguiente esquema muestra, de forma abstracta, los bloques conceptuales en los que se descompondrá la aplicación Java encargada de procesar los eventos del sistema GEO-RBAC:

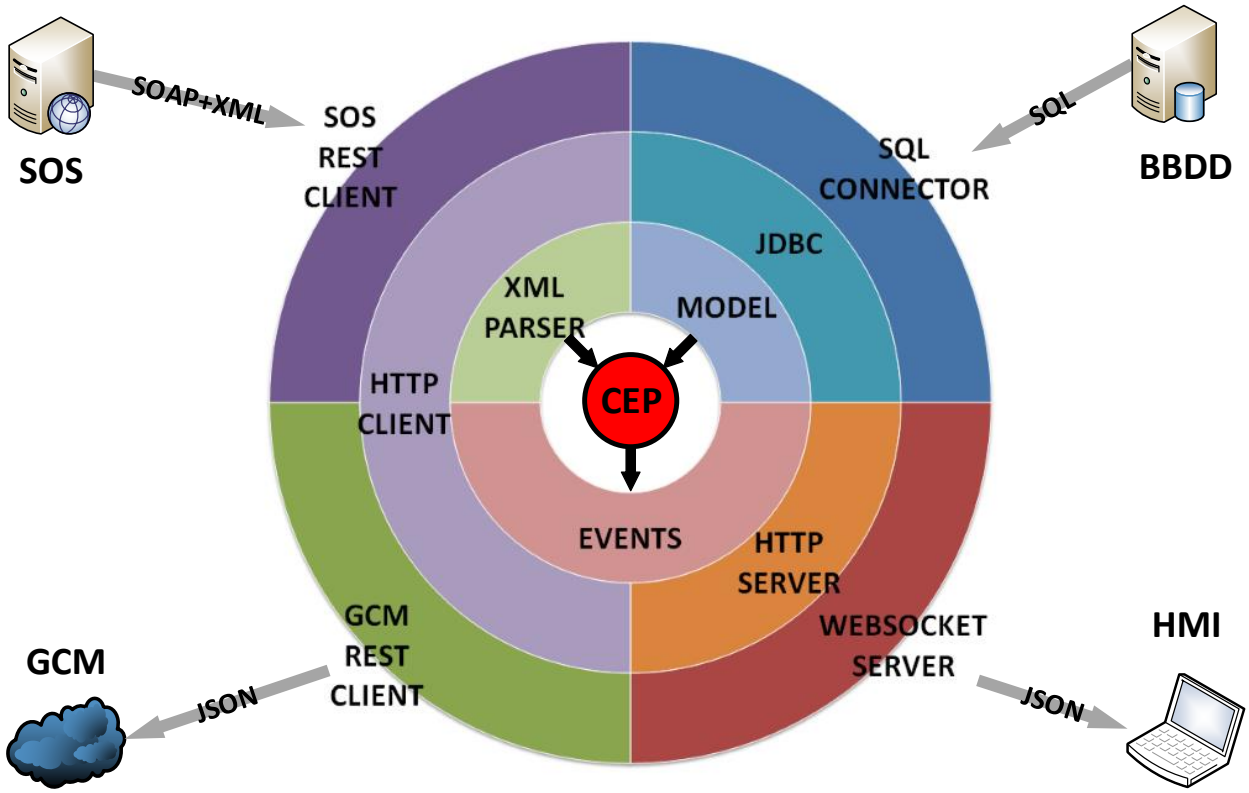


Figura 3.3: Esquema del procesador de eventos

Como se puede observar, la aplicación está dividida en cuatro capas formadas por varios bloques, de forma que los bloques de una capa sirven de

soporte para la siguiente capa. Las funciones que realizará cada bloque serán las siguientes:

Capa 1:

CEP:

Es el bloque principal de la aplicación. Recibe del XML PARSER la posición e identificación del dispositivo y de MODEL el rol y el usuario asociado al dispositivo. Seguidamente contrastará estos datos con las zonas configuradas y generará un evento en caso de que el dispositivo haya entrado o salido de una zona no autorizada.

Capa 2:

XML PARSER:

Este bloque se encargará de recibir el mensaje XML desde el cliente HTTP y buscar dentro del mensaje la posición e identificación del dispositivo para transmitirlo al CEP.

MODEL:

Este bloque realizará las peticiones SQL necesarias para obtener los datos relacionados con el dispositivo que se quiere identificar, con los datos recibidos construirá un objeto que introducirá al CEP.

EVENTS:

Este bloque lo componen los objetos que representan los eventos generados por el CEP, que posteriormente serán introducidos en las salidas que existan.

Capa 3:

JDBC:

Es la librería de Java que permite establecer una conexión mediante un conector a una base de datos relacional y ejecutar sentencias SQL. No depende del sistema de gestión de bases de datos utilizado.

HTTP SERVER:

Es el servidor que gestiona las conexiones HTTP y se encarga de hacer un el "handshake" y "upgrade" para elevar la conexión a una conexión WebSocket.

HTTP CLIENT:

Se encarga de recibir los datos necesarios para realizar conexiones HTTP (al menos el url, método y cuerpo) y establecerlas, devolviendo el resultado de dicha conexión.

Capa 4:

SOS REST CLIENT:

Este bloque utiliza el cliente HTTP para establecer la conexión con el SOS y recibir el mensaje en formato XML+SOAP con la información de los sensores.

SQL CONNECTOR:

Será la librería que se encargará de establecer la conexión con la base de datos. Es específica para el sistema de gestión de bases de datos utilizado.

WEBSOCKET SERVER

Este bloque se encargará de gestionar las conexiones WebSocket establecidas y realizar el intercambio de mensajes en formato JSON.

GCM REST CLIENT

Realiza la conexión con los servidores de Google Cloud Messaging para enviar los eventos que ocurran. El formate de mensajes soportado actualmente es JSON.

De forma transversal a los bloques también existirá un bloque encargado en la configuración del resto de los elementos y otro que realizará las funciones de "logging" necesarias.

El núcleo de la aplicación está compuesto por las capas 1 y 2, de forma que la capa 3 está separada de la capa 2. Ésto permite que se pueda adaptar la aplicación a cualquier otro caso de uso, reemplazando o añadiendo elementos de las capas 3 y 4.

Capítulo 4

Implementación

En este capítulo se especificará el diseño y la estructura de la aplicación siguiendo la arquitectura mostrada en el capítulo anterior, así como las librerías utilizadas mediante diagramas UML. Seguidamente, se mostrará la secuencia de acciones de la aplicación. Finalmente, se describirán brevemente las librerías utilizadas para su desarrollo.

4.1. Diagrama de clases UML

En esta sección se explicará cómo se ha decidido implementar la aplicación siguiendo el esquema de bloques de la arquitectura de la aplicación. Los diagramas se representarán con notación UML.

4.1.1. Diagrama de clases reducido

Con el diagrama de clases reducido se obtendrá una visión de conjunto de la aplicación, en el que agruparemos las clases que formen uno o varios bloques de la arquitectura propuesta:

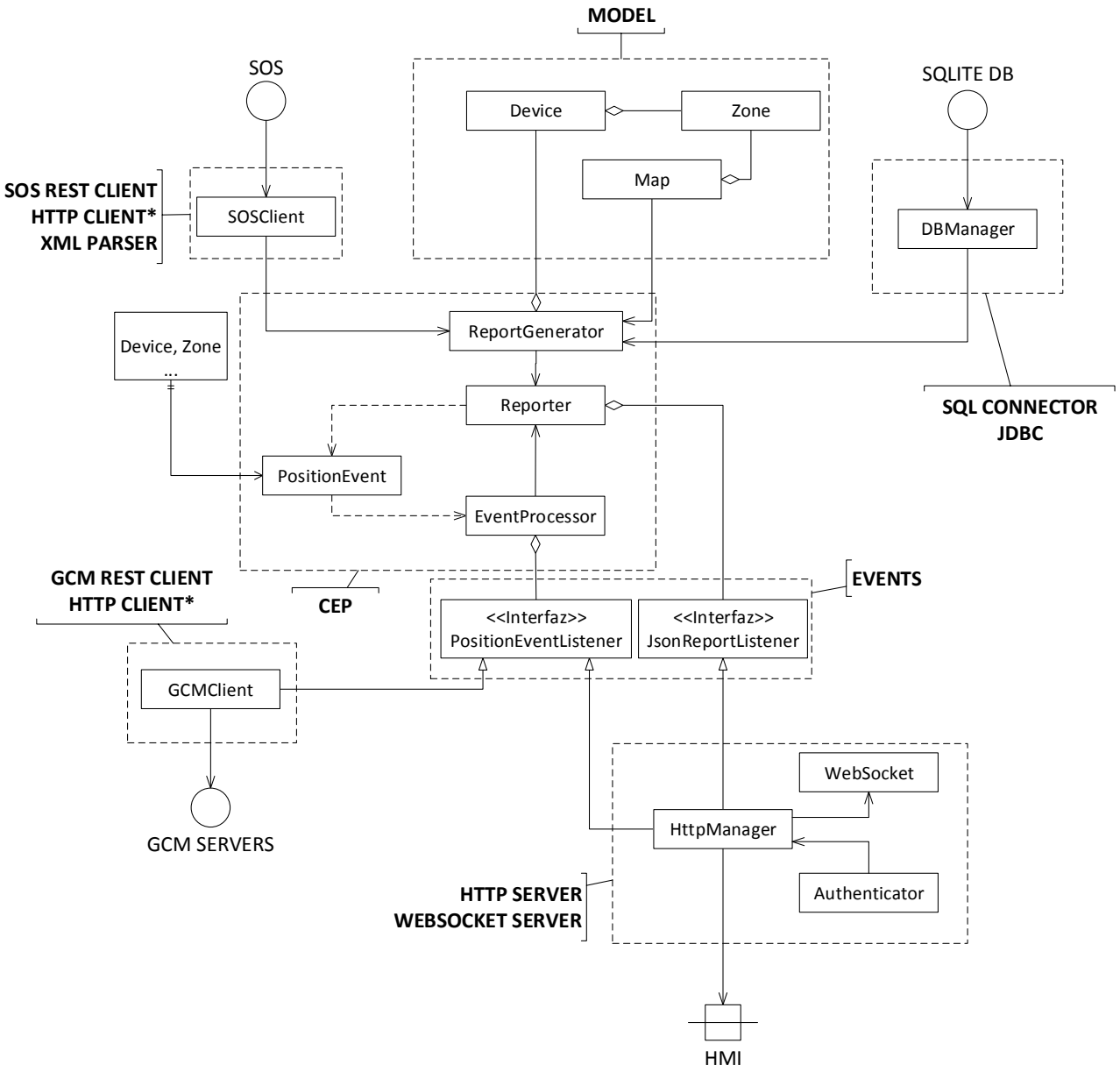


Figura 4.1: Diagrama de clases UML reducido

Se puede observar que algunas clases abarcan varios bloques, se ha decidido realizar de esta manera para simplificar la interacción entre clases y la organización del código.

Los puntos en los que la aplicación se comunica con el exterior se han anotado de la siguiente manera: las conexiones establecidas hacia otros servidores están marcados con una circunferencia, mientras que el puerto que permite conexiones entrantes está marcado con un cuadrado tachado. El sentido de la flecha que une los puntos de conexión con las clases que las manejan indica si la conexión se establece para recibir o enviar datos, aunque conviene recordar que en los protocolos utilizados el intercambio de mensajes es bidireccional.

Tal y como se ha explicado en la arquitectura, la configuración y el logging son comunes en todos los bloques, por tanto, cada clase tendrá acceso a una instancia que permita obtener la configuración de la aplicación y otra instancia que permita realizar el logging de los sucesos que ocurran. Por este motivo, se obviarán en los diagramas representados.

4.1.2. Diagrama de clases UML expandido

A continuación se detallarán, las distintas clases que hemos visto en el diagrama global agrupadas de la siguiente manera:

Entrada de eventos:

Contiene las clases incluidas en los bloques SQL CONNECTOR, JDBC, SOS REST CLIENT, HTTP CLIENT y XML PARSER. Se encarga de recoger los datos de los puntos de entrada y construir los eventos de entrada.

Proceso de eventos:

Contiene las clases del bloque CEP y EVENTS que procesará los eventos de entrada, generará los eventos de salida y los pasará a los elementos que recojan los eventos de salida.

Salida de eventos:

Contiene las clases de los bloques GCM REST CLIENT, HTTP CLIENT, HTTP SERVER y WEBSOCKET SERVER. Su tarea consiste en recoger los eventos de salida y distribuirlos por los puntos de salida.

Explicar que las librerías de terceros tienen un * y las nativas de Java que no son tipos nativos tienen **. También explicar las librerías comunes.

4.1.3. Diagrama de clases UML: Entrada de eventos

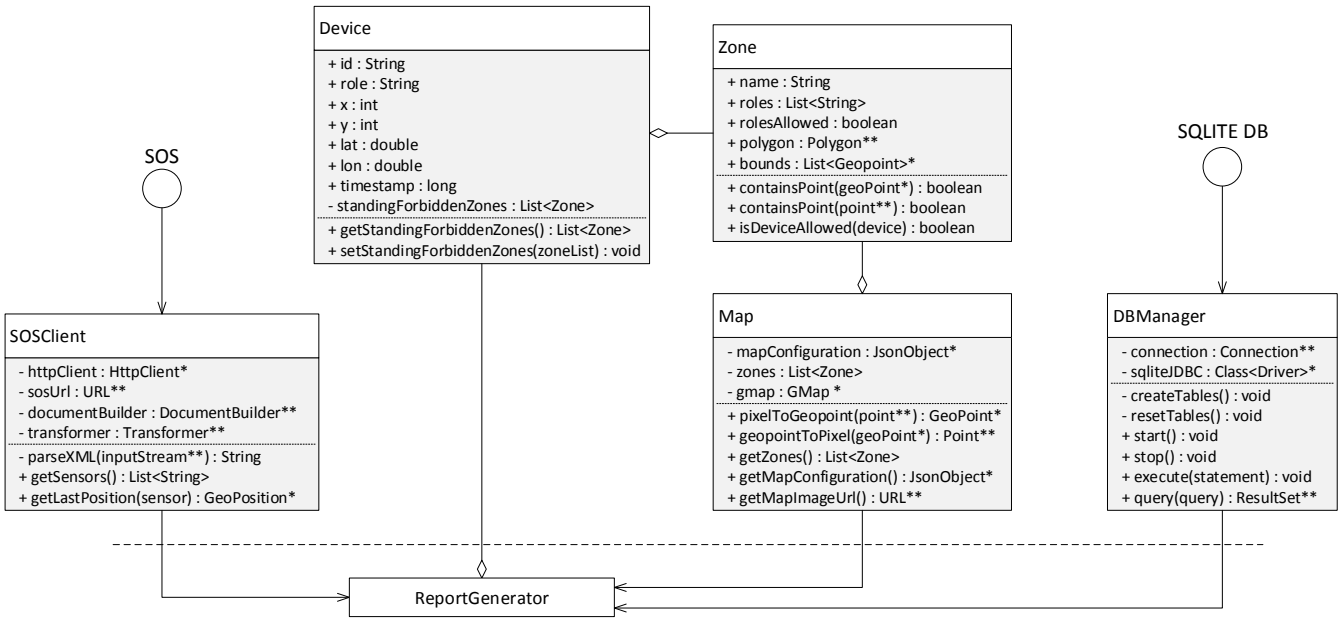


Figura 4.2: Diagrama de clases UML: entrada de eventos

SOSClient:

Su función es comunicarse con el servidor donde esté desplegado el SOS. Para ello, obtiene la URL del servidor del fichero de configuración y realiza la petición utilizando la librería HttpClient 4.4 de Apache. La función "parseXML" transforma la respuesta del servidor en un documento XML y

lo devuelve como una cadena de texto. Dependiendo de la petición realizada, "getSensors" buscará en la cadena de texto de la respuesta del servidor la lista de sensores posibles, mientras que "getLastPosition" obtendrá la última posición del sensor elegido.

DBManager:

Permite ejecutar sentencias SQL a la base de datos de la aplicación. Para ello, el conector del sistema de gestión de bases de datos escogida (en este caso, SQLite) establecerá la conexión a la base de datos. Esta clase ejecutará sus funciones en un hilo propio.

Device:

Contiene los datos que identifican al dispositivo: id, rol, posición relativa en la imagen del mapa (x,y), posición geográfica (lat,lon), una marca de tiempo de la última posición obtenida y una lista de zonas prohibidas en las que se encuentra. Como veremos más adelante, esta lista es imprescindible para decidir si el dispositivo entra o sale de una zona en la que no está autorizado.

Zone:

Contiene los datos de una zona específica del mapa: nombre de la zona, polígono que determina el perímetro (en coordenadas relativas al mapa y coordenadas geográficas), y una lista de los roles que están (o no) autorizados. Que los roles especificados estén autorizados o no lo determina el atributo "rolesAllowed". Además, esta clase tendrá las funciones necesarias para determinar si un dispositivo se encuentra en la zona y si está autorizado. Para determinar si el dispositivo se encuentra dentro de la zona se utilizará el conocido algoritmo PIP (Point In Polygon) que decide si un punto está dentro o fuera dependiendo de si el número de lados cruzados hasta un punto exterior es par o impar.

Map:

Esta clase contendrá la configuración del mapa actual desde un archivo JSON y cargará las zonas establecidas y hará uso de la librería auxiliar desarrollada para obtener la imagen y realizar la conversión de coordenadas geográficas a puntos en la imagen.

4.1.4. Diagrama de clases UML: Proceso de eventos

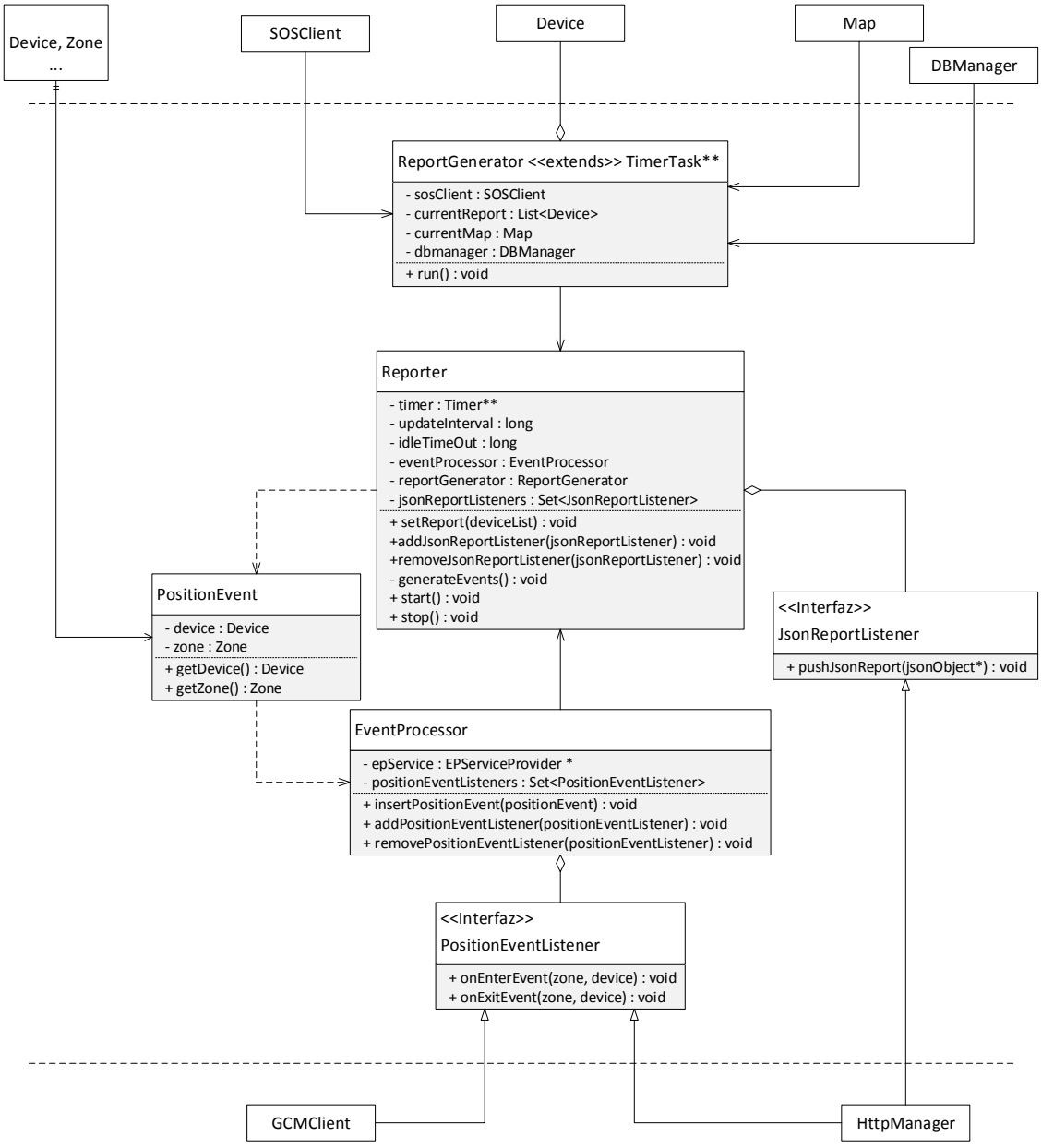


Figura 4.3: Diagrama de clases UML: proceso de eventos

Reporter:

Es la clase principal del proceso de eventos. Se encarga de recibir los reportes que contienen la última posición conocida de los dispositivos, decide si están activos o no (dependiendo de si el último movimiento de un dispositivo ocurrió fuera o dentro del margen establecido por "idleTimeOut"), y genera los eventos entrada combinando cada dispositivo con cada zona del mapa. De forma opcional, podrá enviar el reporte con la última posición de los dispositivos a los puntos de salida que implementen la interfaz JsonReportListener. Inicializará los hilos de las clases ReportGenerator y EventProcessor.

ReportGenerator:

Esta clase conecta los bloques de entrada de datos con el proceso de eventos. Se ejecuta en un hilo propio y su función es realizar periódicamente un reporte (una lista de los dispositivos existentes con su última ubicación). Para ello combinará los datos de los sensores que ofrece la clase SOSClient con los datos obtenidos de la base de datos a través de DBManager para crear la lista de dispositivos. También puede utilizar la clase Map para convertir la posición de coordenadas geográficas a coordenadas relativas a la imagen del mapa. Esto último es opcional, ya que el algoritmo PIP para decidir si el dispositivo se encuentra o no en una zona funciona también con coordenadas geográficas.

PostionEvent:

Son los eventos de entrada generados por Reporter y que serán introducidos en EventProcessor. Contiene un dispositivo (con su última posición) y una zona, existe uno por cada combinación de dispositivo/zona.

EventProcessor:

Esta clase configura la librería que realiza la gestión y el procesado de eventos mediante la clase EPServiceProvider. Le introduce las sentencias necesarias para distinguir qué eventos de salida surgirán a partir de los eventos de entrada. Para ello contiene los métodos necesarios para insertar

eventos de entrada así como para que se registren las clases que implementen la interfaz PositionEventListener, necesaria para recibir los eventos de salida. Esta clase también se ejecutará en un hilo propio.

JsonReportListener:

Es la interfaz que deberán implementar las clases que quieran recibir los reportes con la posición actualizada de cada dispositivo. El reporte que recibirán estará ya en formato JSON.

PositionEventListener:

Esta interfaz la implementan las clases tengan que recibir los eventos de salida. Habrá dos tipos de eventos de salida: un evento que informa que el dispositivo ha entrado a una zona no autorizada (enterEvent) y otro evento que informa que ha salido de una zona no autorizada (exitEvent).

4.1.5. Diagrama de clases UML: Salida de eventos

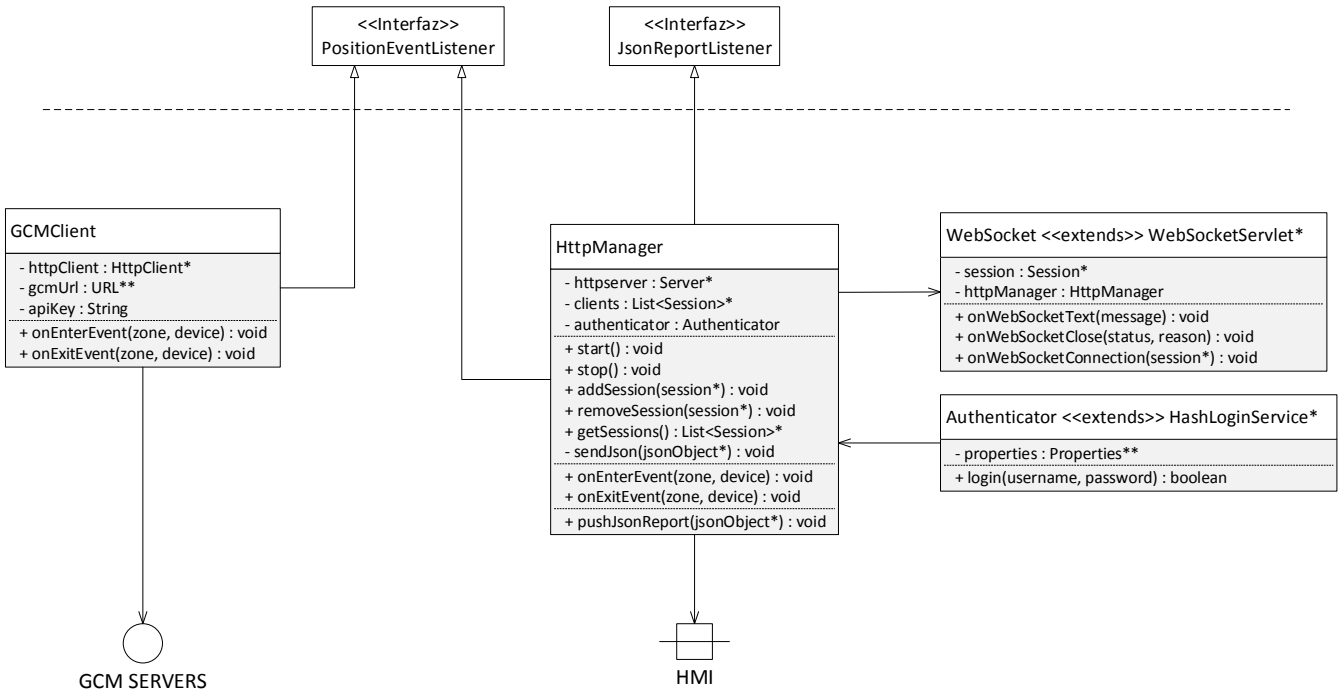


Figura 4.4: Diagrama de clases UML: salida de eventos

GCMClient:

Implementa la interfaz PositionEventListener para recibir los eventos de entrada/salida de una zona no autorizada por parte de un dispositivo. Su función principal será comunicarse con los servidores de Google Cloud Messaging y retransmitir el evento mediante una petición POST, siendo el contenido un mensaje en formato JSON con los parámetros necesarios para que sea aceptado.

HttpManager:

Esta clase implementará tanto PositionEventListener como JsonReportListener para poder recibir tanto eventos como reportes. Inicializará el servidor estableciendo el puerto y el certificado SSL, además establece que la clase

WebSocket se encargará de manejar las conexiones que pidan "upgrade" a protocolo websocket, y que la clase Authenticator tendrá que autorizar el acceso a la conexión.

WebSocket:

Cuando se establezca una conexión mediante el protocolo websocket, esta clase tomará el control de la conexión y realizará el intercambio de mensajes. Para cada conexión, el servidor ejecutará esta clase en un hilo auxiliar distinto, que terminará cuando se cierre la conexión.

Authenticator:

A partir de un fichero de configuración, esta clase autorizará mediante usuario/contraseña que la conexión tiene acceso para recibir los eventos y reportes.

4.2. Ciclo de vida de la aplicación

En esta sección se tratará de explicar la secuencia de acciones que realiza la aplicación, divididas por hilos, pues cada uno realizará las acciones en paralelo. En el siguiente esquema se muestran las acciones principales que ejecutarán los hilos:

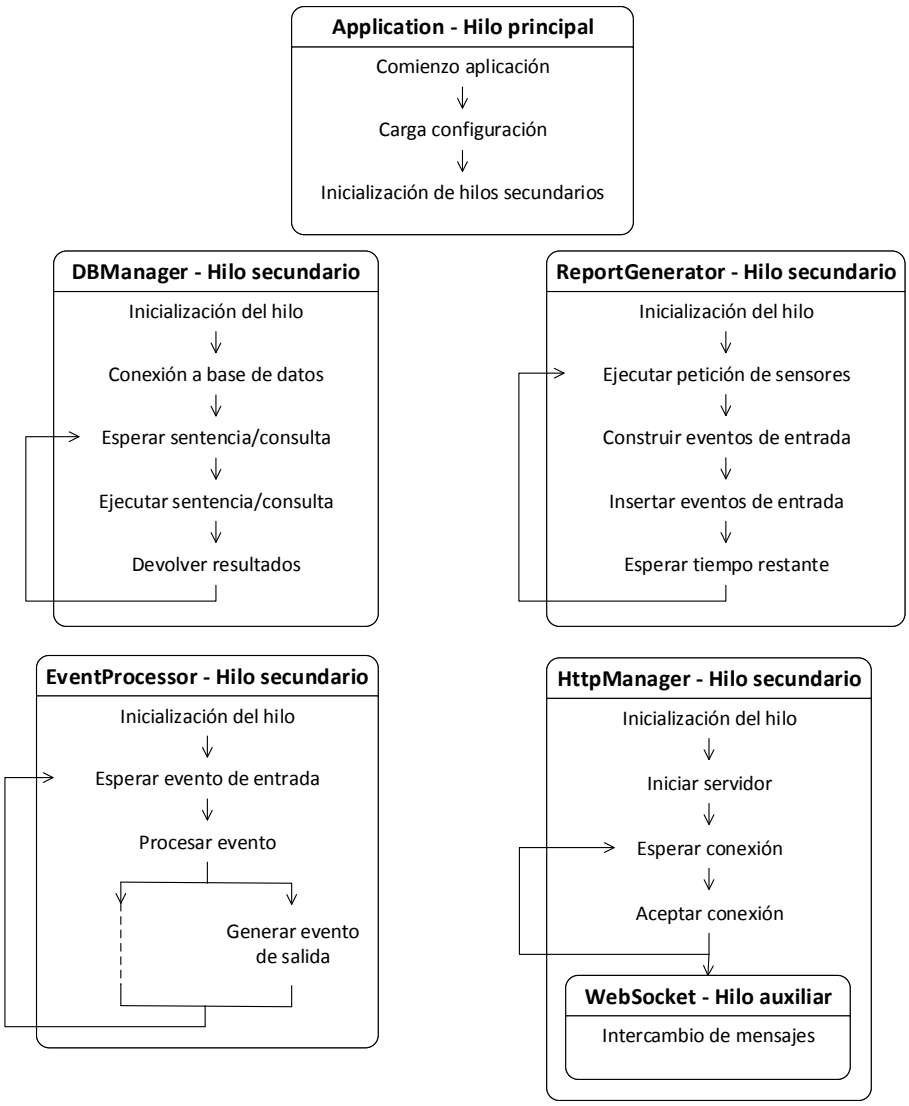


Figura 4.5: Secuencia de acciones de cada hilo

Application:

En el hilo principal comienza a ejecutarse la aplicación, su primera acción será iniciar el sistema de logging cargando el archivo de configuración de logging "log4j2.xml" que se encontrará en el Classpath. A continuación, tratará de buscar el archivo de configuración de la aplicación, cuya ruta recibe por entrada, si no existe buscará el archivo de configuración por defecto, si tampoco existe o el archivo tiene un formato incompatible a los archivos de propiedades de Java abortará la ejecución. Ésta configuración será un singleton accesible por todas las demás clases de la aplicación. Finalmente, iniciará el resto de los hilos en el orden que se encuentra en el esquema.

DBManager:

Una vez inicializado el hilo, se conectará a la base de datos especificado en el archivo de configuración. Su función a partir de entonces será esperar a que cualquier otra clase le pida la ejecución de una sentencia o consulta, que realizará y devolverá los resultados. A la hora de implementarlo, se decidió que se pudiese proceder de dos formas: de forma síncrona, mediante el uso de un "lock" que no permita a las demás clases acceder mientras esté ejecutando una petición, o de forma asíncrona, encolando la petición y avisando mediante un "callback" cuando estén listos los resultados.

ReportGenerator:

Este hilo, una vez ha sido inicializado, pedirá la posición de los sensores y posteriormente interaccionará con el hilo DBManager para ejecutar las peticiones necesarias a la base de datos para construir los eventos de entrada a partir de la posición de los sensores. Una vez construidos interaccionará con el hilo EventProcessor para pasar los eventos de entrada creados. Posteriormente, esperará un tiempo especificado en el fichero de configuración para realizar la siguiente llamada.

La tarea que ejecuta este hilo, se planifica mediante la clase Timer de Java. Ésta clase nos da la opción de planificar tareas de distintas formas, en este caso, se contemplaron dos formas: esperando un tiempo asegurado después de la realización de la tarea o ejecutando la siguiente tarea tras transcurrir

un intervalo de tiempo desde el inicio de la tarea anterior, aunque ésta todavía no haya acabado. Ésta última opción quedó descartada ya que si el número de dispositivos es muy alto deja de responder adecuadamente.

EventProcessor:

Tras configurar la librería del CEP con las sentencias que generan los eventos de salida a partir de los eventos de entrada, el hilo es inicializado y ejecutado por la librería, que nos da la opción de insertar los eventos de entrada y de recibir los de salida.

HttpManager:

Este hilo contendrá la ejecución del servidor HTTP. Esperará conexiones entrantes y las aceptará si el servidor no está congestionado. Si la conexión pide el "upgrade" a WebSocket, se iniciará un hilo auxiliar para manejar la conexión establecida. Cada hilo auxiliar interaccionará con EventProcessor para recibir los eventos de salida. Cuando la conexión se cierre el hilo terminará.

4.3. Librerías utilizadas

esper-4.10.0: Esta librería se encarga de encolar, gestionar y tratar los eventos introducidos. Se configura mediante unas sentencias escritas en su lenguaje propio y éstas decidirán si se generan eventos de salida a partir de los eventos de entrada.

log4j-2.1: Librería utilizada para hacer logging de lo sucedido.

gson-2.2.4: Librería para escribir y leer mensajes en formato JSON.

javax-servlet-3.0.0 y jetty-all-9.2.1: Ambas utilizadas para implementar el servidor HTTP embebido en la aplicación.

sqlite-jdbc-9.7.15: Librería que contiene el conector necesario para establecer conexión con bases de datos SQLite.

georbac-maptool: Es la única librería que no es de terceros, y la explicaremos en la sección de aplicaciones auxiliares.

Capítulo 5

Aplicaciones auxiliares

Paralelamente al desarrollo de la aplicación ha sido necesario crear una serie de aplicaciones para facilitar el desarrollo, ejecución y prueba de la aplicación principal.

5.1. Creador de mapas

Está escrita en Java y se creó a partir de la librería georbac-maptool, que también fue desarrollada.

Esta librería fue creada con el objetivo de poder obtener imágenes estáticas de los servidores de Google Maps, aunque también funciona con MapServer si se activa el acceso a "tiles" (subdivisiones del mapa). Una vez obtenidos la imágen del mapa se pueden realizar conversiones de posiciones geográficas (latitud/longitud) a píxels de la imágen (x,y) y viceversa. También se pueden realizar cálculos de distancia.

Para realizar esta conversión, se utiliza la Proyección Transversa de Mercator (desarrollado por el Cuerpo de Ingenieros del Ejército de los Estados Unidos en 1940 a partir de la proyección de estudiada por Gerardus Mercator en 1569). En esta proyección los meridianos se proyectan sobre el plano con una separación proporcional a la del modelo, mientras que los paralelos se van separando a medida que se alejan del Ecuador, de forma que son infinitas en los extremos, aunque la región terrestre sólo se representa entre los paralelos 84ºN y 80ºS. El modelo que se utiliza para representar la tierra es WGS 84 (World Geodetic System 84), que modela el globo terráqueo como un elipsoide.

Con la Proyección Transversa de Mercator convertimos las coordenadas

geográficas (latitud/longitud) a coordenadas en el sistema UTM (Universal Transverse Mercator), este sistema de coordenadas es cartesiano, y se puede manipular para poder convertirlo en píxels del mapa. Para realizar esta conversión, utilizamos el centro del mapa solicitado como punto de referencia.

Para solicitar la imagen del mapa es necesario introducir dos parámetros de entrada: el centro del mapa en coordenadas geográficas y las dimensiones en píxels de la imagen del mismo. Por tanto, se puede utilizar el centro de la imagen del mapa (anchura/2,altura/2) y la conversión UTM de las coordenadas geográficas como punto de referencia para el resto de puntos del mapa, pues al tratarse de un sistema cartesiano la conversión es escalar y se puede realizar mediante una matriz de transformación.

Una vez diseñada la librería se desarrolló una aplicación a partir de ella que es el creador de mapas para el sistema GEO-RBAC realizado. En la aplicación se puede escoger de forma visual el mapa sobre el cual se desea trabajar y sobre él crear y configurar las zonas protegidas.

Por cada zona creada se puede especificar una lista de roles y escoger si estos roles estarán autorizados o no en la zona. Una vez configurado el mapa, se genera un archivo JSON con la configuración que será utilizada en la configuración del procesador de eventos.

En la siguiente imagen se pueden ver unas capturas de pantalla de la aplicación:

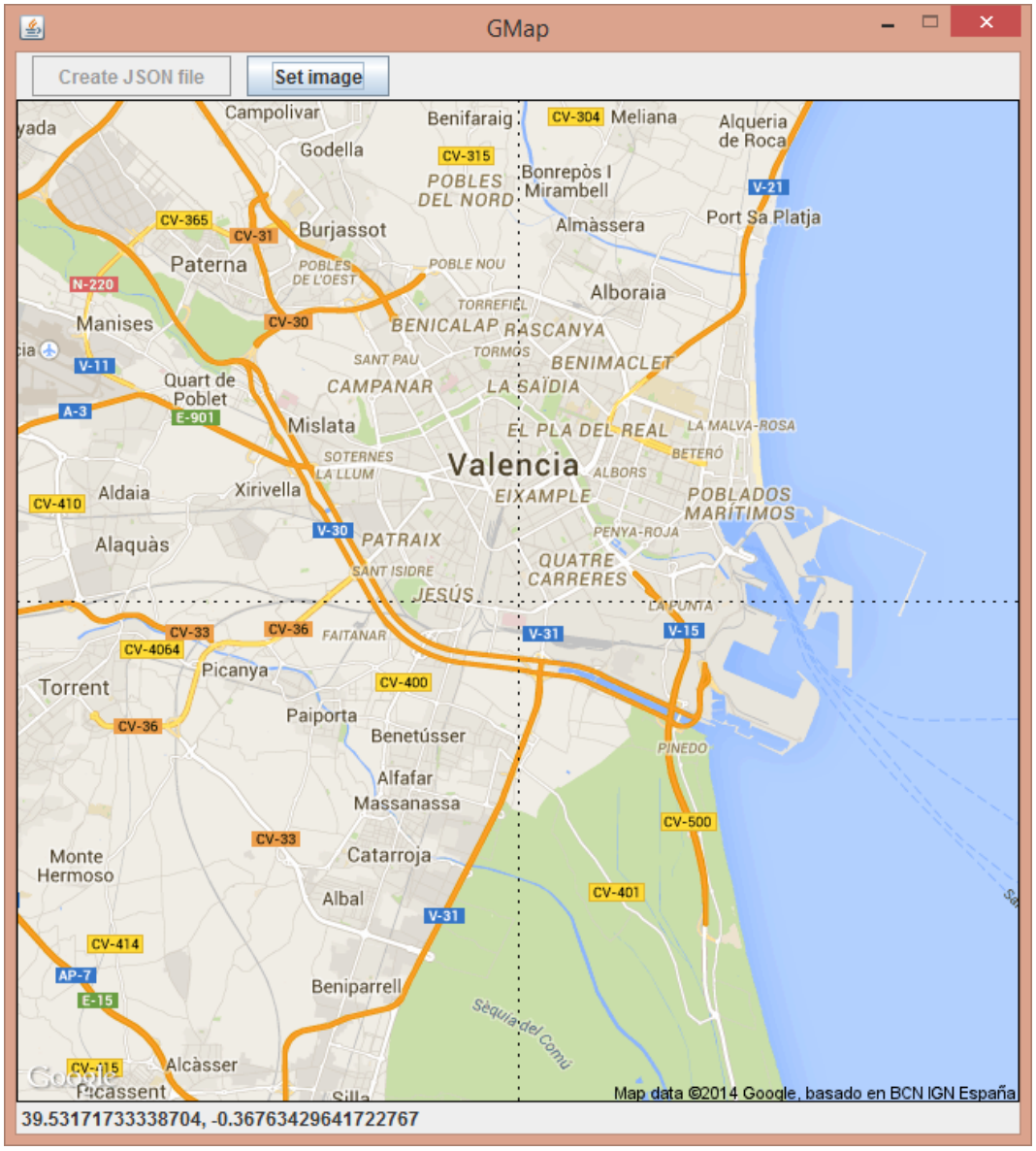


Figura 5.1: Creador de mapas: Navegación

Con el ratón se puede realizar zoom hacia dentro y hacia fuera, al igual que desplazarse por el mapa. La posición geográfica que aparece abajo la izquierda indica la posición del puntero del ratón.

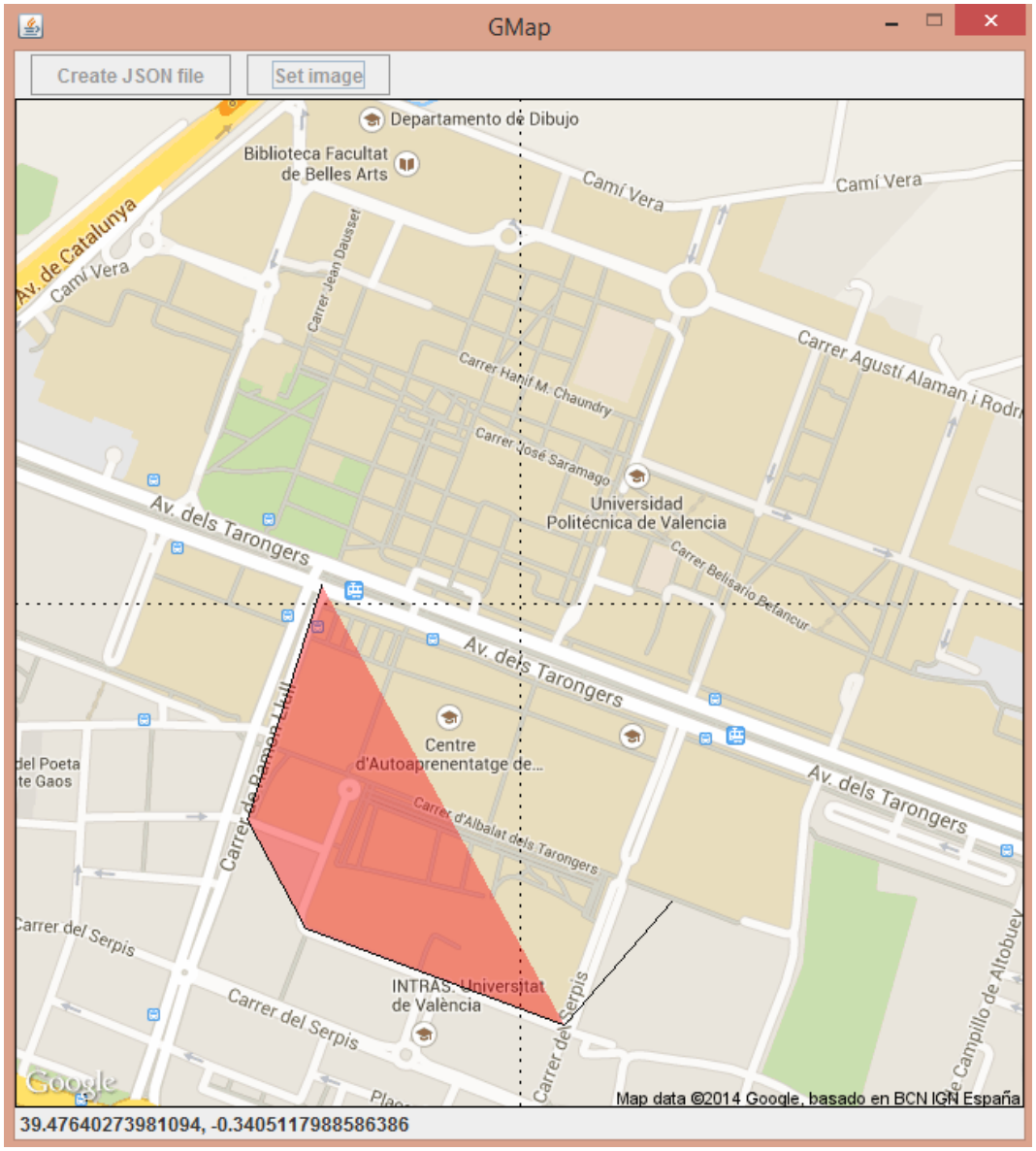


Figura 5.2: Creador de mapas: Creación de una zona

Se pueden crear múltiples zonas, indicando los vértices que forman el perímetro del polígono.

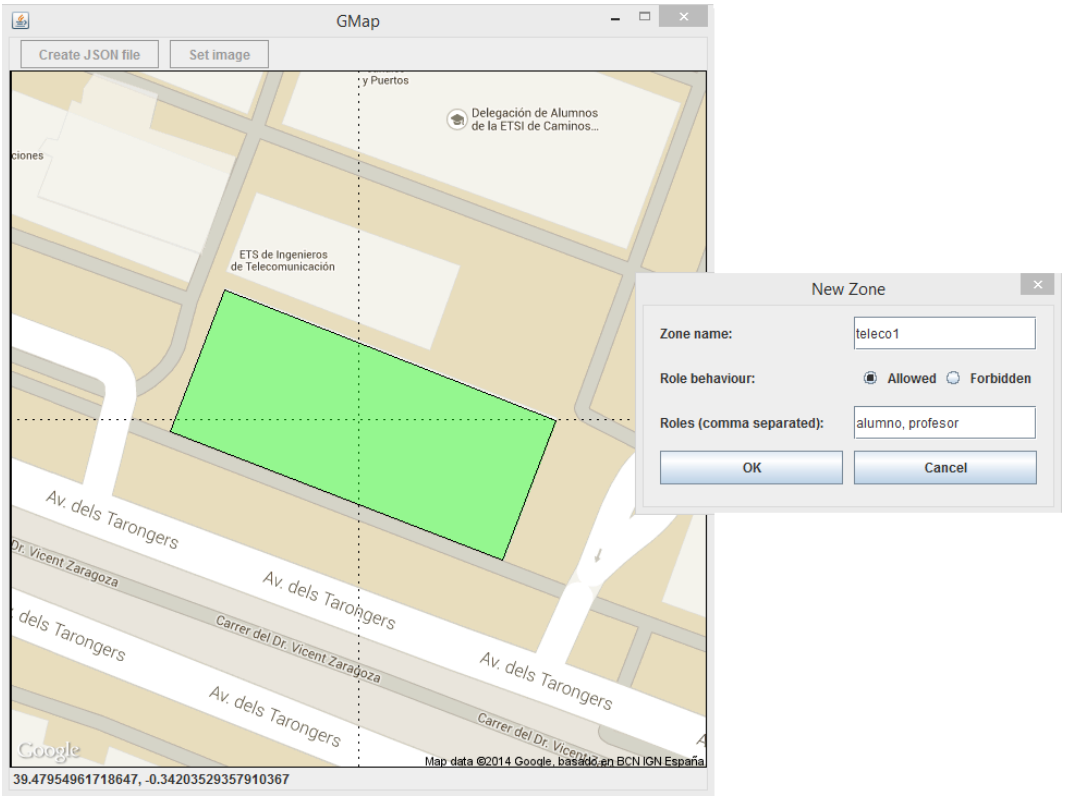


Figura 5.3: Creador de mapas: Configuración de una zona

Una vez configuradas las zonas se puede generar el archivo JSON con la configuración del mapa y de las zonas.

Aunque el API de Google Maps permite realizar las conversiones cartográficas necesarias, se prefirió implementar la opción de realizar la conversión dentro del sistema para no depender de una conexión a Internet una vez el sistema está configurado y en marcha.

5.2. Simulador de SOS

A la hora de desarrollar el procesador de eventos, era necesario ir realizado pruebas intermedias para asegurar el funcionamiento. Para tal fin se creó un simulador de SOS, simulando mediante una interfaz de Java la clase que posteriormente se encargaría de conectarse al SOS.

El simulador toma los dispositivos registrados en la base de datos del procesador de eventos y cada vez que era necesario dar la última posición de cada

dispositivo, se utilizaba el algoritmo Random Waypoint para que se asemejara a una situación real.

El algoritmo Random Waypoint es un modelo de movilidad que en lugar de generar de forma aleatoria la siguiente posición, toma en consideración el destino, velocidad, aceleración y dirección a la hora de calcular la siguiente posición.

5.3. Simulador de dispositivos

El simulador de dipositivos está escrito en Javascript, HTML5 y CSS. Se desarrolló con el mismo objetivo que el simulador de SOS, pero en lugar de tomar posiciones aleatorias se puede configurar gráficamente el camino que se va a recorrer. Fue necesaria desarrollarla para poder realizar pruebas concretas y simular un recorrido real.

El simulador de dispositivos es una aplicación Web, por tanto puede ser ejecutada en cualquier navegador que soporte HTML5 (concretamente, el API "canvas"). Esta aplicación realizará peticiones HTTP (POST) al SOS para insertar las posiciones, pero eso solamente será posible realizarlo siempre y cuando el dominio (incluido el puerto) es el mismo en la aplicación origen y en la aplicación destino. Sin embargo, hoy en día la mayoría de los navegadores Web permiten realizar peticiones HTTP a servidores de orígenes distintos siempre y cuando tengan habilitado lo que se conoce como CORS.

CORS (Cross-Origin Resource Sharing) es un mecanismo que ha de activarse en el servidor, se implementa mediante la cabecera "Access-Control-Allow-Origin" en las respuestas HTTP. En esta cabecera se establecen los dominios autorizados para realizar peticiones AJAX (se puede utilizar un wildcard para permitir cualquier dominio). Por suerte, en el SOS de 52North es posible habilitar esta opción.

En la siguiente imagen se puede observar el funcionamiento de esta aplicación Web:

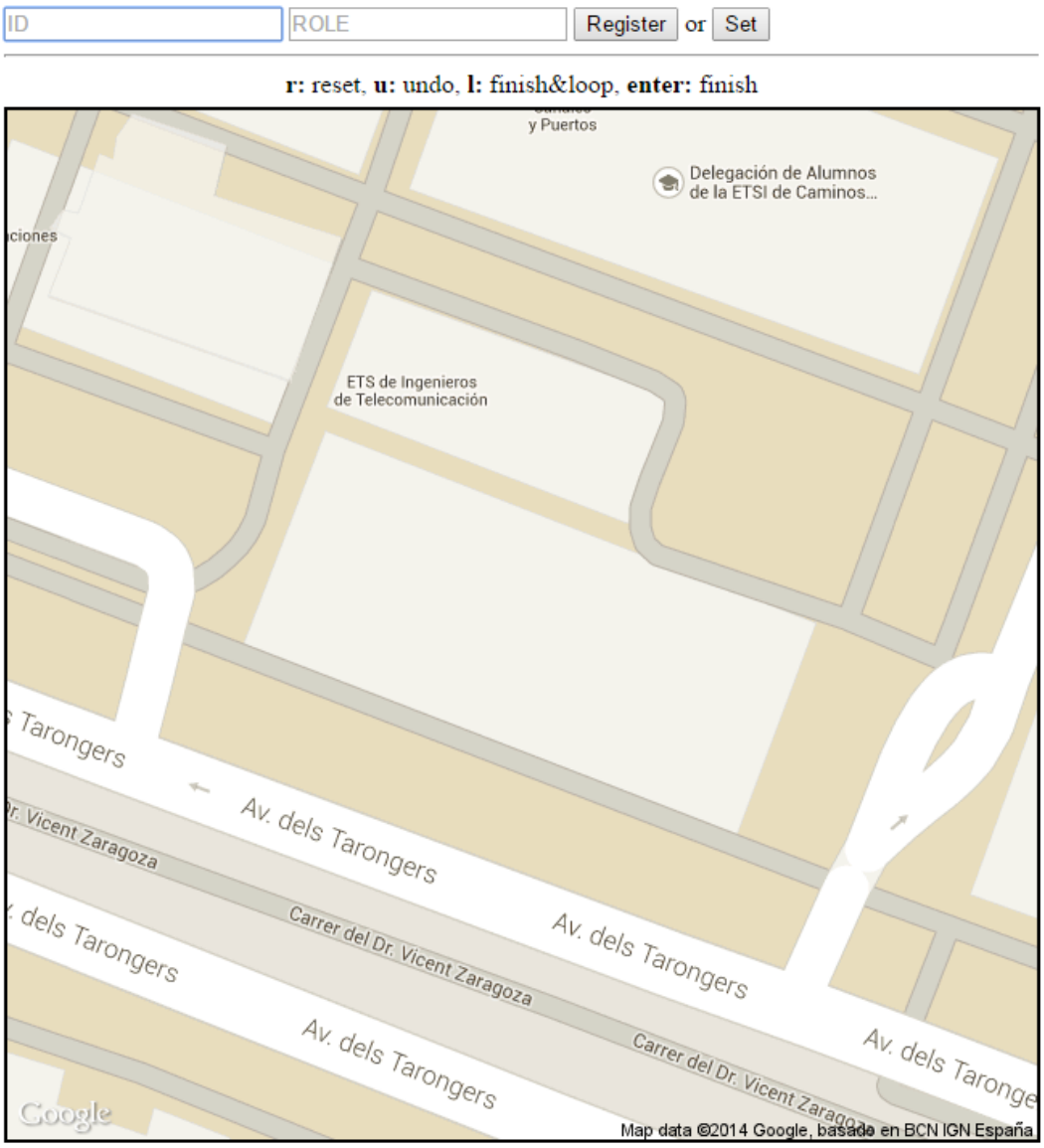


Figura 5.4: Simulador de eventos: Estado inicial

Si el procesador de eventos está configurado y en funcionamiento, el simulador le pide el mapa que se está utilizando. Una vez se tiene el mapa, se puede introducir la ID y el ROL del dispositivo que se quiere simular.

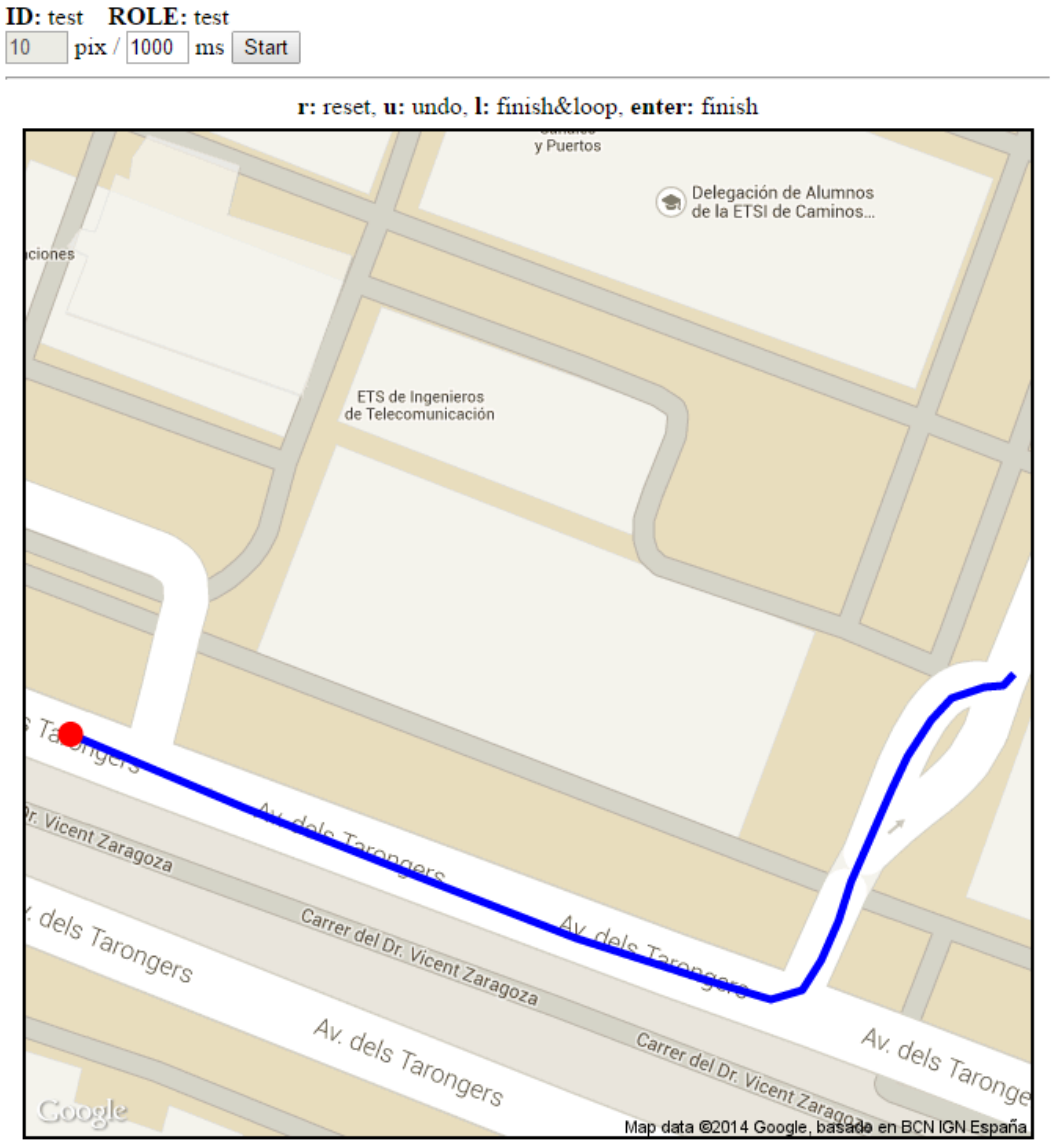


Figura 5.5: Simulador de eventos: Dibujo del camino

Se pueden dibujar los puntos del camino indicando los puntos de los segmentos en los que se descompone el camino que se quiere dibujar. Mediante los controles que se pueden observar se puede empezar de nuevo el camino (reset), deshacer el último segmento (undo), terminar el camino en bucle (tecla "L"), o terminar el camino y no realizar bucle (tecla "Enter").

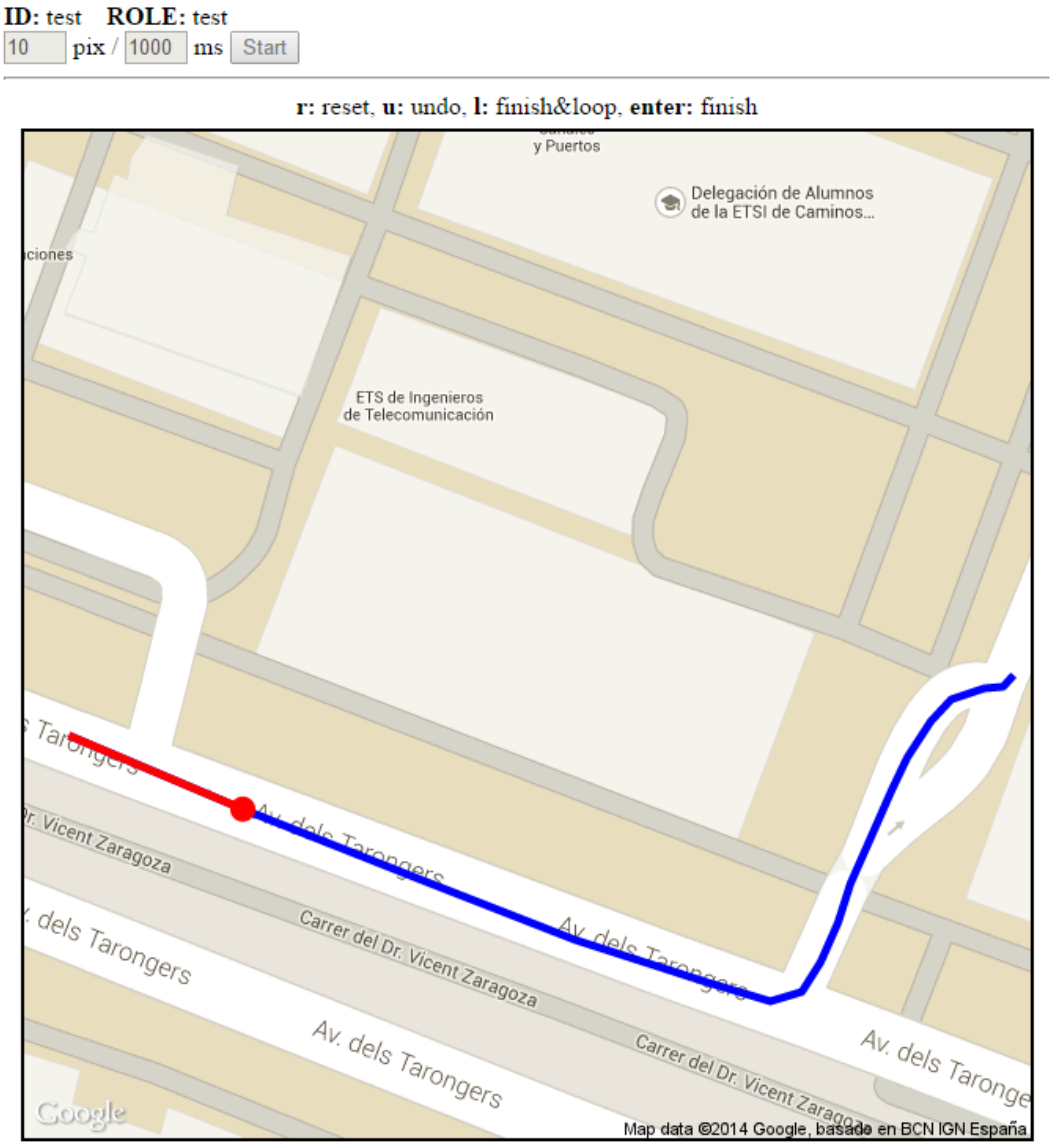


Figura 5.6: Simulador de eventos: Inicio de la simulación

Una vez dibujado el camino se puede configurar el número de píxels que se recorrerán cada intervalo de tiempo. El intervalo de tiempo, en milisegundos, también es configurable. Una vez se configuran los parámetros se puede empezar la simulación, que recorrerá el camino trazado.

Para enviar las posiciones al SOS, se deben enviar como coordenadas geográficas, por tanto, se ha tenido que implementar la librería georbac-maptool (explicada en la aplicación del creador de mapas) en Javascript, y realizar la

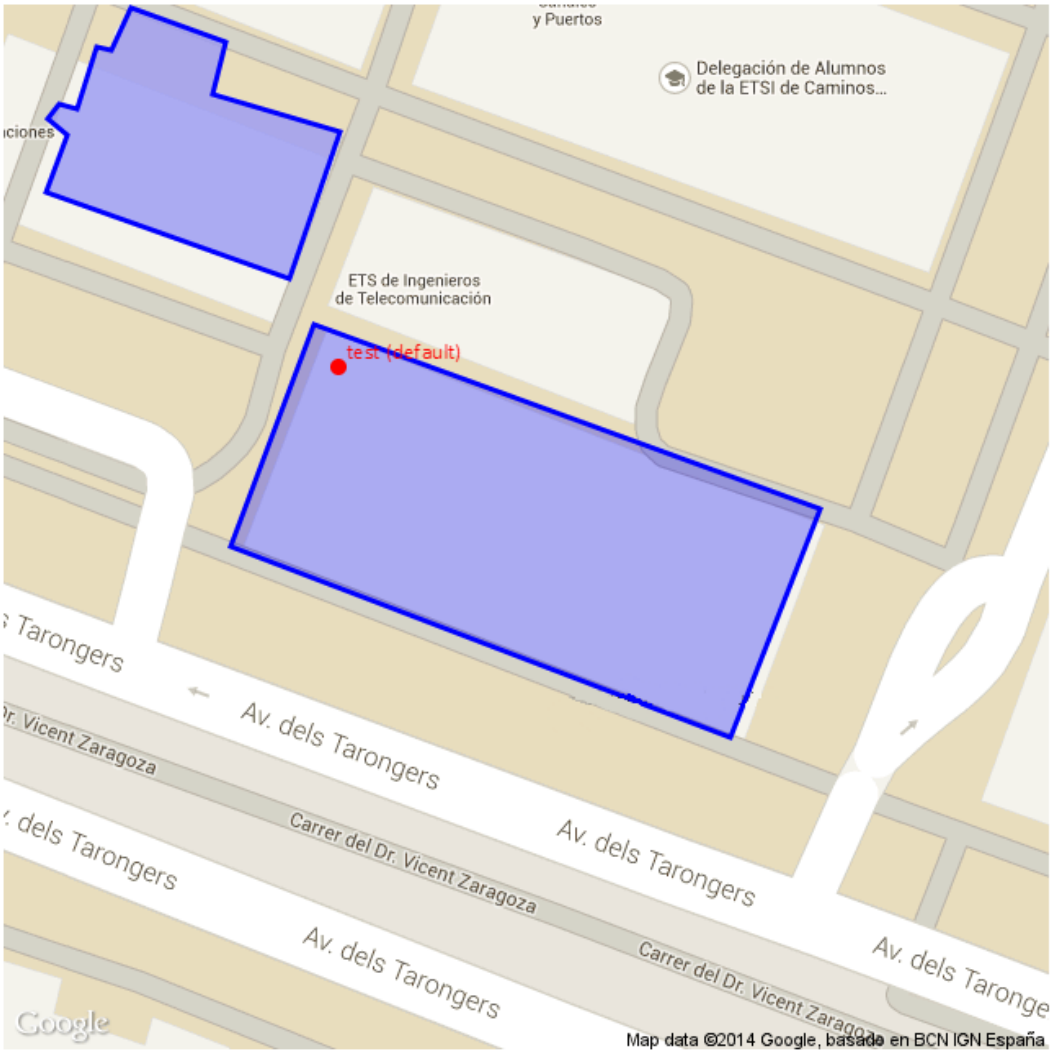
conversión de píxels en la imagen a coordenadas en latitud/longitud.

5.4. Visualizador de eventos

Esta es la última aplicación desarrollada para poder probar el funcionamiento del procesador de eventos. Se trata de una aplicación Web escrita en Javascript, HTML5 y CSS cuya finalidad es simular la interfaz gráfica de gestión, y así comprobar que se está realizando correctamente el control de acceso.

Para ello se conecta mediante WebSocket al procesador de eventos y recibe la configuración del mapa, los reportes, y los eventos de salida. Como en el simulador de dispositivos, esta aplicación Web ha de ejecutarse en navegadores que soporten HTML5 (el api "canvas" y "websocket").

En la siguiente imagen se puede ver la interfaz de la aplicación, donde se puede ver el histórico de eventos en la parte inferior:



20:34:54.88 07-12-2014 - test (default) entered teleco2

20:34:57.86 07-12-2014 - test (default) exited teleco2

20:35:02.70 07-12-2014 - test (default) entered teleco1

Figura 5.7: Simulador de eventos: Inicio de la simulación

Se puede observar la fecha y hora en el que se produce un evento, el nombre (ID) del dispositivo, entre paréntesis el rol del dispositivo y el tipo de evento ocurrido.

Capítulo 6

Pruebas de ejecución y resultados

En este capítulo vamos a describir el proceso que se ha llevado a cabo para realizar las pruebas necesarias al software desarrollado. Primero se explicará el despliegue de las aplicaciones, y a continuación se describirán los diferentes escenarios que se han escogido para realizar las pruebas. En cada escenario se explicarán los datos de entrada utilizados y se mostrarán los resultados obtenidos.

En todos los escenarios menos en el primero, el software utilizado para realizar las pruebas y las medidas es Gatling. Variando los parámetros de entrada obtendremos los diferentes escenarios. Para realizar las pruebas automáticas se habilitó un servicio de registro de sensores automático, para no tener que introducir de forma "manual" cada sensor en el sistema.

De los distintos resultados que se verán a partir del escenario 2, obviaremos los resultados del servidor de registro, pues no es relevante a la hora de valorar la respuesta del sistema, ya que en una situación real el registro sería esporádico comparado con el envío de posiciones.

6.1. Despliegue de las aplicaciones

Para ejecutar las pruebas y verificar el funcionamiento correcto, es necesario desplegar, como mínimo, el SOS y la aplicación que se ha desarrollado. El SOS que se utilizó, como ya se ha comentado, es el desarrollado por 52North en su última versión(4.1.0). Este SOS se distribuye como un "Web Applica-

tion Archive" por lo que se debe desplegar sobre un contenedor de Servlets. El contenedor que se decidió utilizar fue Jetty en su última versión estable (versión 9). A parte de un contenedor de Servlets, el SOS necesita también un sistema de gestión de bases de datos compatible, en este caso, se escogió MySQL también en su última versión estable (Community Edition 5.6.21).

Tanto la aplicación desarrollada como el SOS elegido se ejecutan sobre una máquina virtual de Java, por tanto, también es necesario instalar el Java Runtime Environment que sea capaz de ejecutar aplicaciones compiladas en la versión 1.7 del compilador de Java. Finalmente, para instalar la aplicación sólo es necesario copiar el archivo JAR compilado y los archivos de configuración necesarios en un directorio cualquiera.

Aunque la aplicación ha sido desarrollada para que pueda ser instalada en un servidor distinto al servidor que contenga el SOS, en las pruebas que se han realizado se encuentran ambas en el mismo servidor junto con el sistema de gestión de bases de datos.

Para ejecutar las pruebas, se utilizaron algunas aplicaciones auxiliares comentadas en el capítulo anterior: el creador de mapas, el simulador de dispositivos y el visualizador de eventos. Además, se utilizó Gatling (en su última versión estable, 2.0.3) para realizar pruebas de estrés. Estas aplicaciones se ejecutarán en máquinas externas al servidor, para que no afecten a la hora de realizar las pruebas. Ninguna de estas aplicaciones necesita ser instalada en el sistema operativo, ya que el creador de mapas desarrollado está compilado en un único JAR ejecutable, Gatling se distribuye como una aplicación portable y tanto el simulador de dispositivos como el visualizador de eventos se ejecutan en un navegador Web (en las pruebas, se ha escogido Chrome).

6.2. Configuración común

Todos los escenarios compartirán una misma configuración, que será sencilla pero suficiente para asegurar que cubre los casos posibles (estar o no autorizado en una zona).

En la siguiente imagen se muestran el mapa y las dos zonas que se van a utilizar para realizar las pruebas:

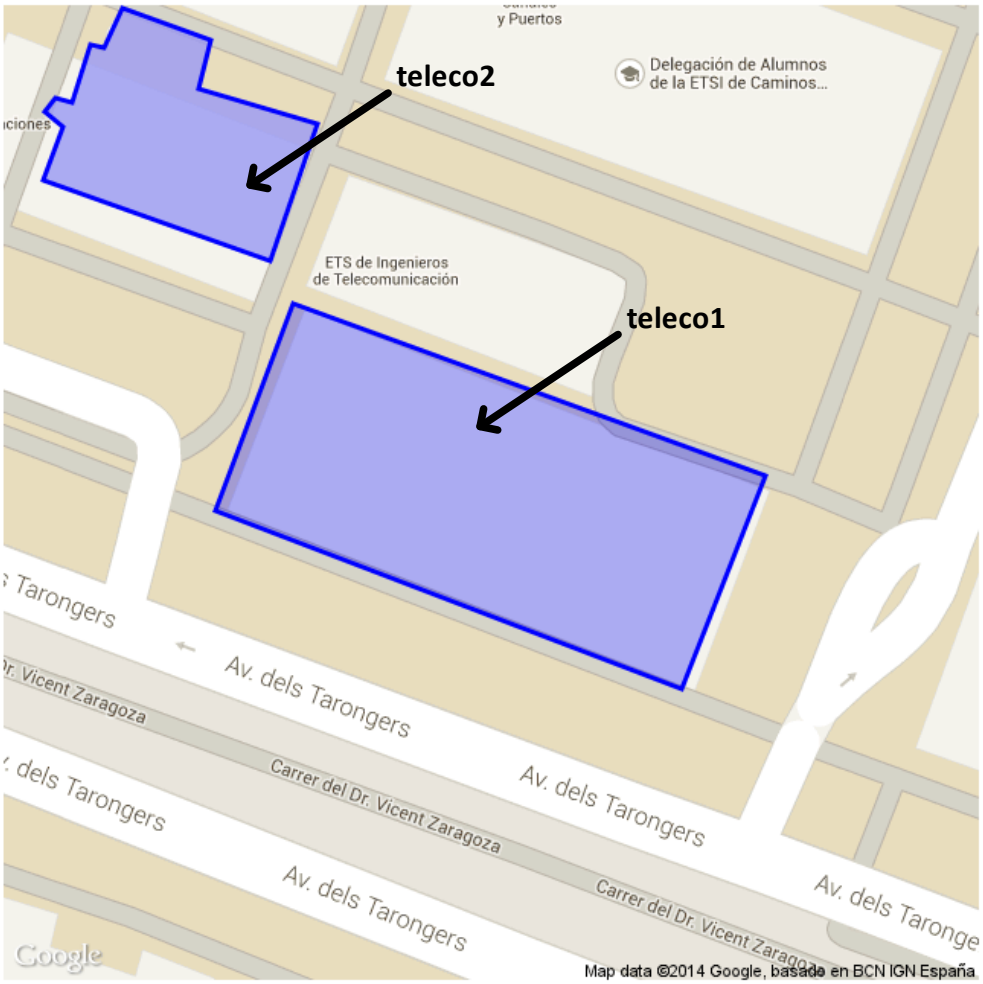


Figura 6.1: Mapa utilizado para las pruebas

En ambas zonas los roles que están permitidos a acceder son “alumno” y “profesor”.

6.3. Escenario 1

El primer escenario es el más sencillo, utilizaremos como entrada un solo dispositivo con un camino trazado previamente. En este escenario se trata de verificar que el sistema funciona correctamente y que se realiza el control de acceso de forma correcta.

Se insertarán tres dispositivos distintos: sujeto1, sujeto2 y sujeto3. Los dos primeros tendrán, respectivamente, el rol “alumno” y “profesor”, por lo que tendrán permitido el acceso a ambas zonas. El tercer sujeto tiene el rol “visitante”, y no tendrá acceso a ninguna.

6.3.1. Datos de entrada

En las siguientes imágenes se mostrarán el camino trazado para cada uno de los sujetos, la configuración del simulador de dispositivos será la misma para los tres: cada segundo se recorrerán 10 píxels de la imagen.

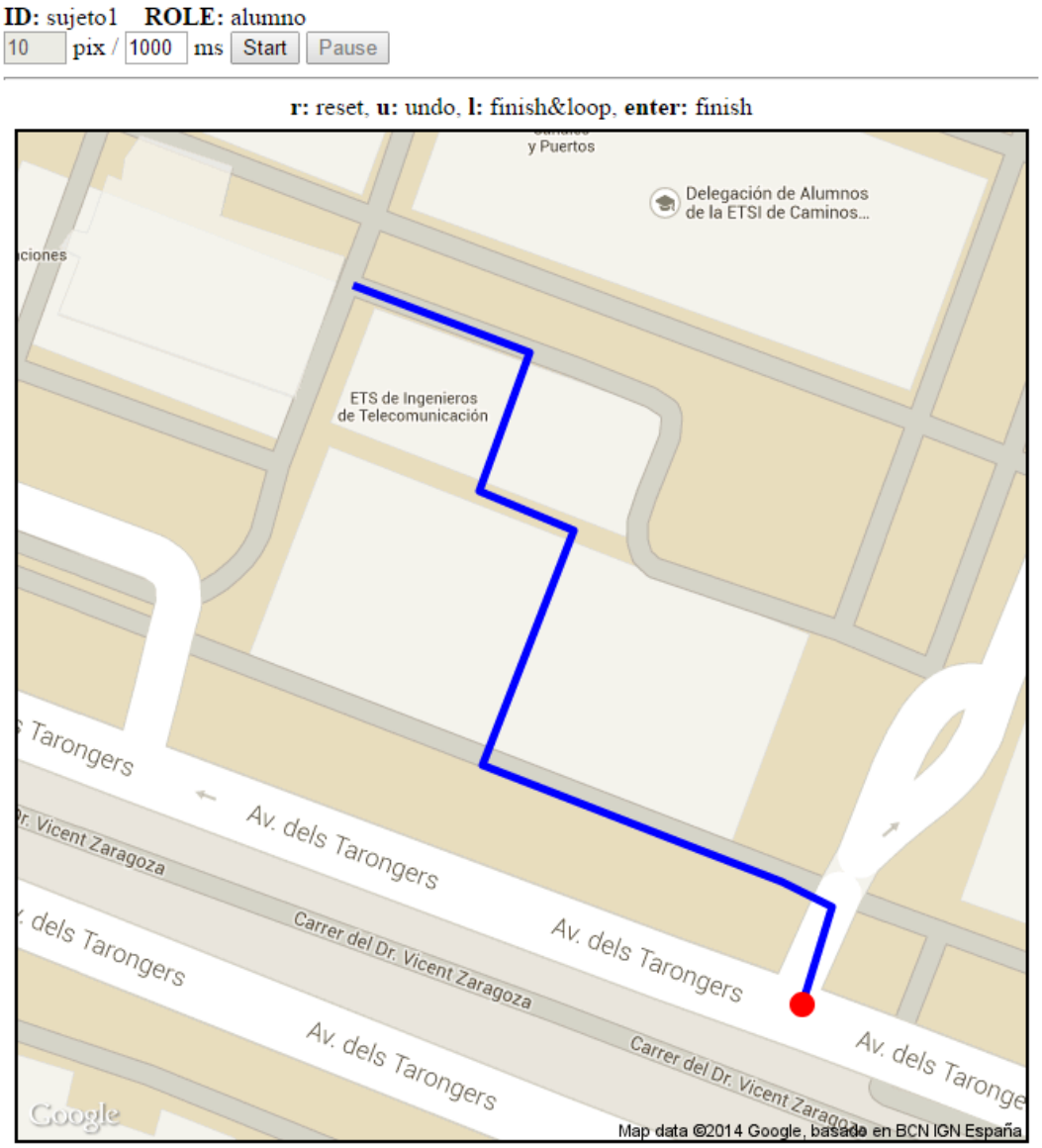


Figura 6.2: Escenario 1: Camino del sujeto 1

Figura 6.3: Escenario 1: Camino del sujeto 2

Figura 6.4: Escenario 1: Camino del sujeto 3

6.3.2. Resultados

Una vez trazados los tres caminos se iniciarán los simuladores a la vez. Las siguientes dos imágenes muestran el estado del visualizador de eventos más o menos en la mitad de la simulación y al final:

Figura 6.5: Escenario 1: Simulación (intermedio)

Se puede observar que el sujeto1 y el sujeto2 han podido entrar a una zona restringida sin disparar ningún evento, además el sujeto2 (profesor) ha cruzado

la zona teleco2 y no ha disparado ningún evento. Sin embargo el sujeto3 ha entrado a una zona no autorizada y el visualizador ha recogido el evento de salida producido por el procesador de eventos.

Figura 6.6: Escenario 1: Simulación (final)

Al final de la simulación se puede ver que los únicos eventos disparados son producidos por el sujeto no autorizado, el sujeto3, y por tanto se está realizando correctamente el control de acceso.

6.4. Escenario 2

6.4.1. Datos de entrada

En este escenario se introducirán 10 usuarios durante 17 segundos (un usuario cada 1.7 segundos), cada uno enviando diez posiciones a intervalos regulares de un segundo. Podemos observar en la siguiente imagen la gráfica de usuarios activos resultante:

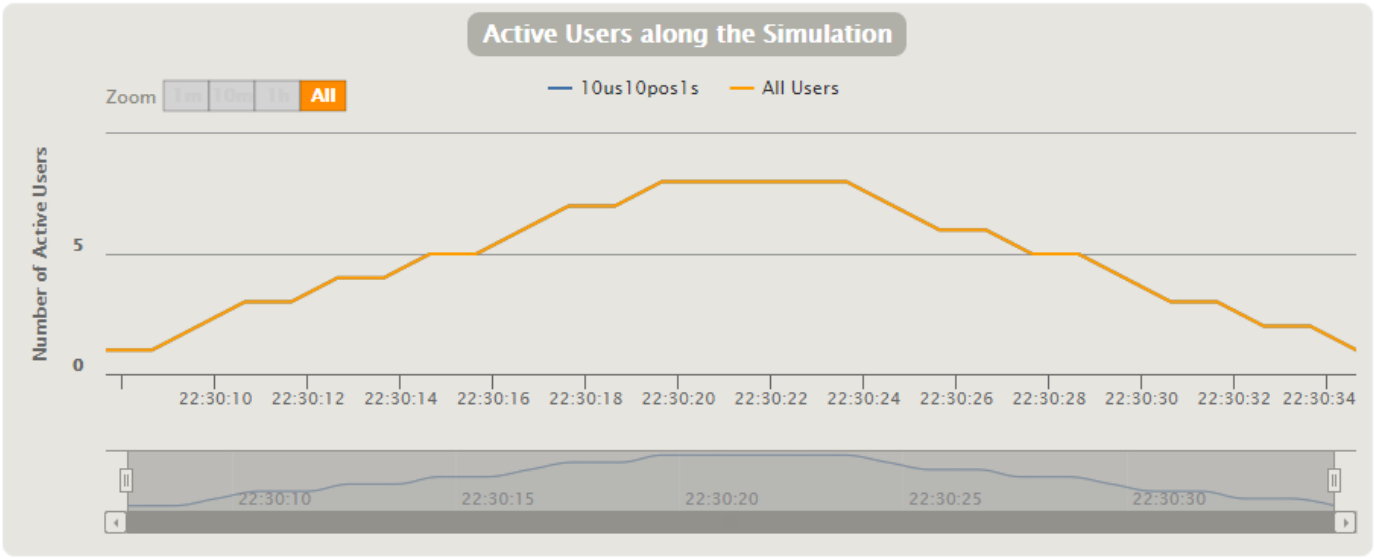


Figura 6.7: Escenario 2: Usuarios activos

Se puede observar que la simulación durará, efectivamente, 27 segundos (el último usuario entra en el segundo 17 y permanece activo durante 10 segundos) y el pico de usuarios activos simultáneamente es de 8, durante los segundos 12 y 16 de la simulación.

6.4.2. Resultados

Estadísticos

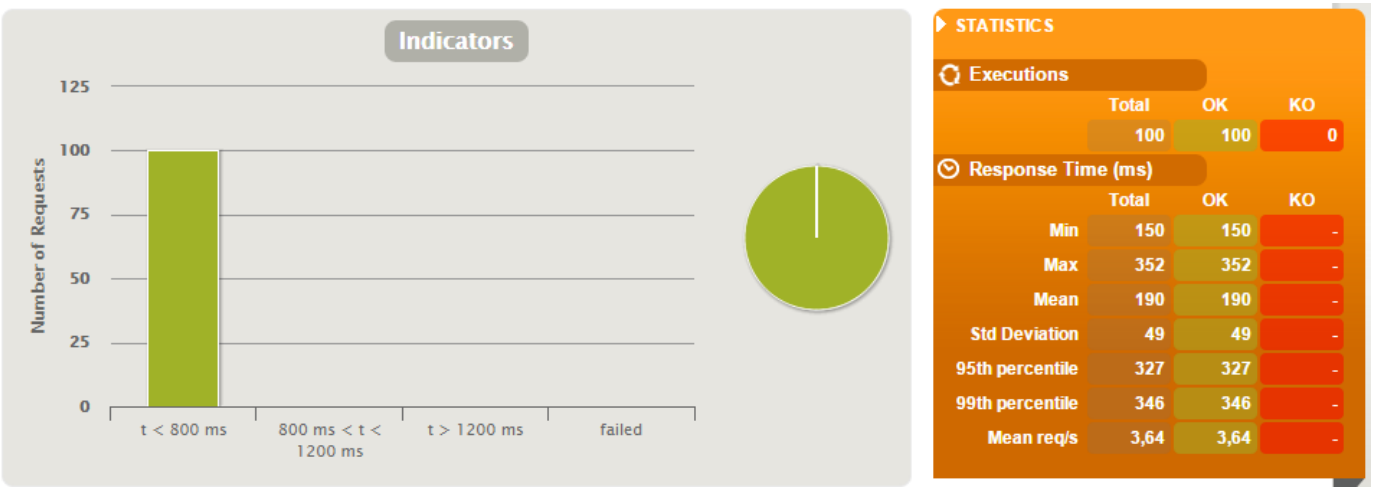


Figura 6.8: Escenario 2: Estadísticos

Se puede observar que de todas las peticiones realizadas ninguna ha fallado, de los 100 envíos de posiciones, el mínimo tiempo de respuesta ha sido de 150ms, mientras que el máximo ha sido de 352ms. La media de respuesta es de 190ms, un valor bastante bueno.

Número de peticiones y respuestas por segundo

Figura 6.9: Escenario 2: Número de peticiones y respuestas por segundo

Como es lógico, tanto el número de peticiones y respuestas sigue la evolución del número de usuarios activos de forma simultánea. Ambos tienen una forma parecida, modificada por el tiempo de respuesta que ha sufrido cada petición. Si hubiese alguna petición/respuesta saldría marcado en rojo, pero en esta prueba no ha habido ningún fallo.

Tiempo de respuesta del sistema

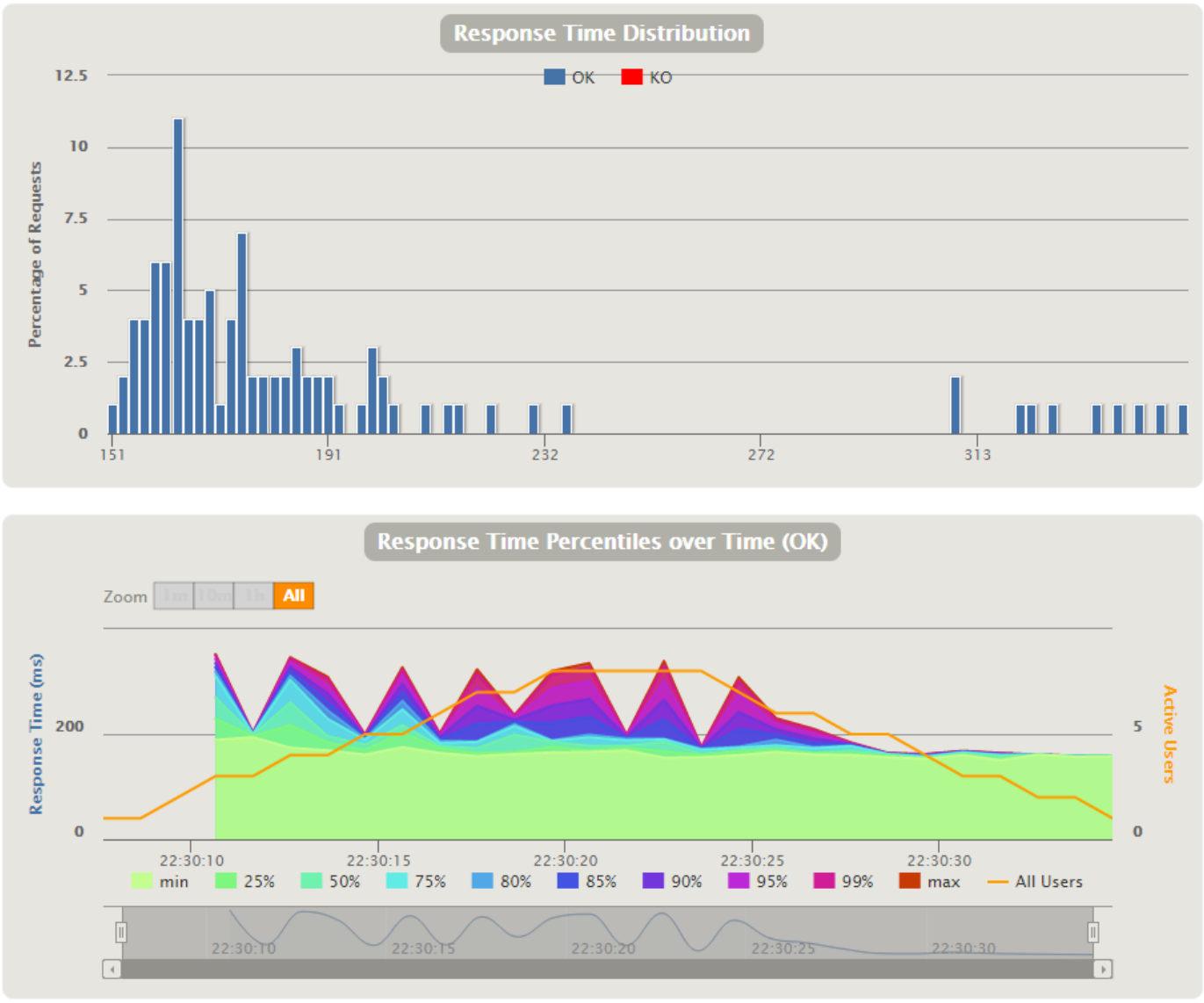


Figura 6.10: Escenario 2: Distribución y percentiles del tiempo de respuesta

En la primera gráfica se puede observar el porcentaje de peticiones que ha sufrido cierto tiempo de respuesta. Hay dos zonas claramente distinguibles: la primera, entre los 151ms y 236ms, la segunda, entre los 309ms y 351ms. En la segunda gráfica se muestra la evolución del tiempo de respuesta del sistema durante el tiempo y en comparación con los usuarios activos, se puede ver como al final de la simulación los tiempos de respuesta descienden y se estabilizan.

6.5. Escenario 3

6.5.1. Datos de entrada

En este escenario se introducirán 100 usuarios durante 30 segundos (un usuario cada 0.3 segundos), cada uno enviando diez posiciones a intervalos regulares de un segundo. Podemos observar en la siguiente imagen la gráfica de usuarios activos resultante:

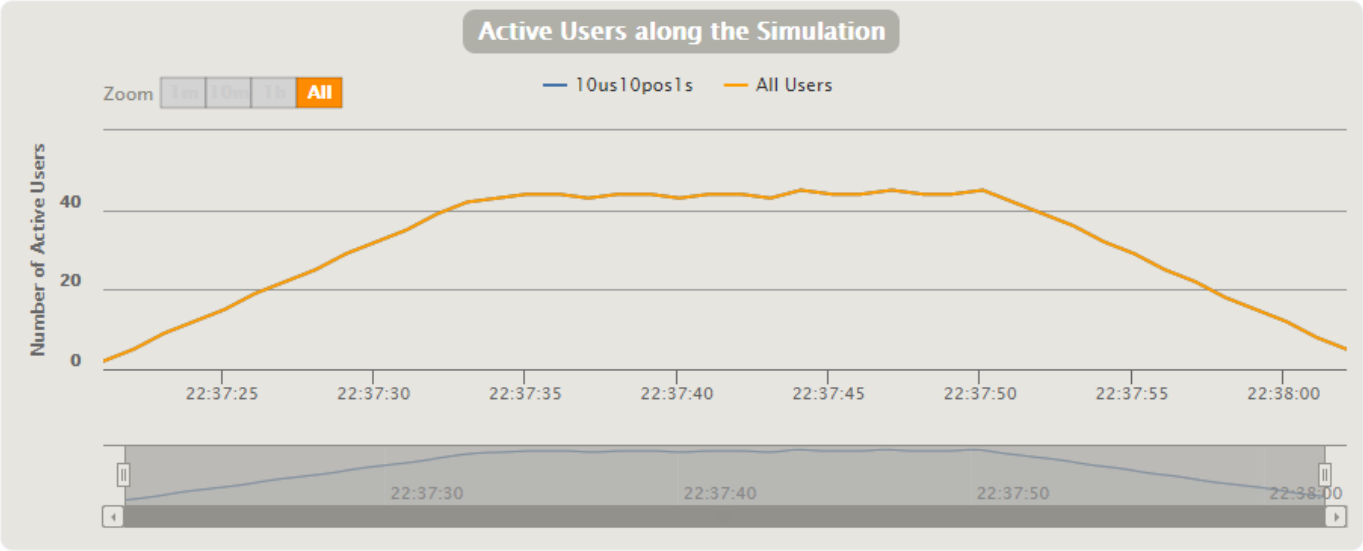


Figura 6.11: Escenario 3: Usuarios activos

La simulación dura 40 segundos más o menos, el pico de usuarios de forma simultánea son 45, aunque se puede observar que en la parte central de la simulación, durante 30 segundos más o menos, hay más de 40 usuarios activos.

6.5.2. Resultados

Estadísticos

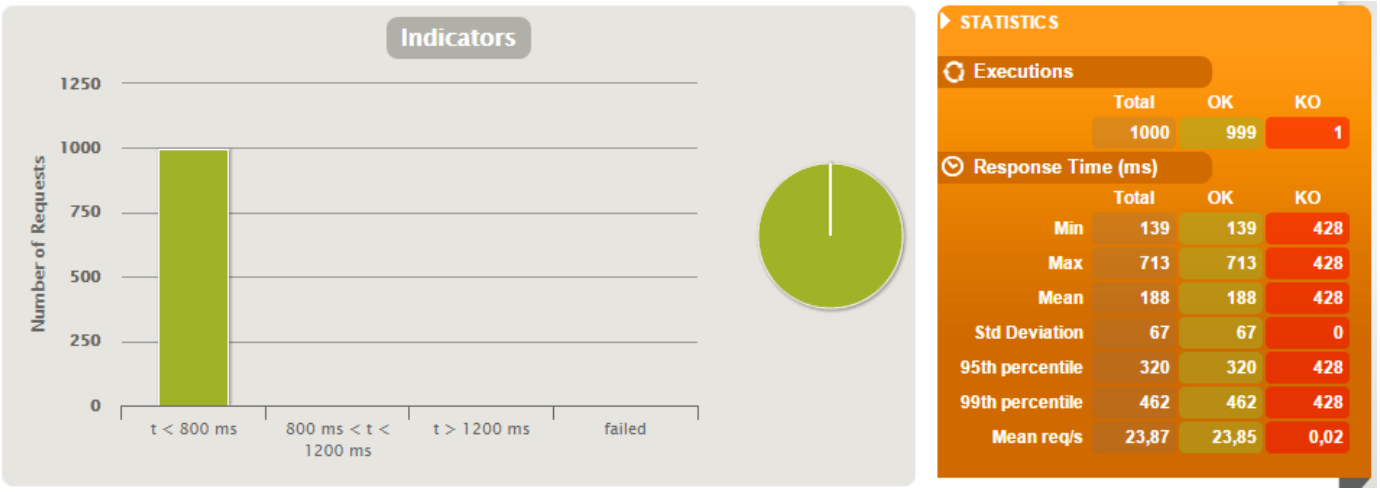


Figura 6.12: Escenario 3: Estadísticos

De las 1000 peticiones que se han realizado, una de ellas ha fallado (aunque no se muestra, el error que devolvió el servidor fue el error 500, un error genérico para indicar un fallo interno del servidor) pero el resto han sido respondidas correctamente. El valor del tiempo de respuesta mínimo y máximo son 139 y 713 milisegundos respectivamente, mientras que la media es de 188 milisegundos. Se puede observar que el 95 % de las respuestas está por debajo de 320 milisegundos.

Número de peticiones y respuestas por segundo

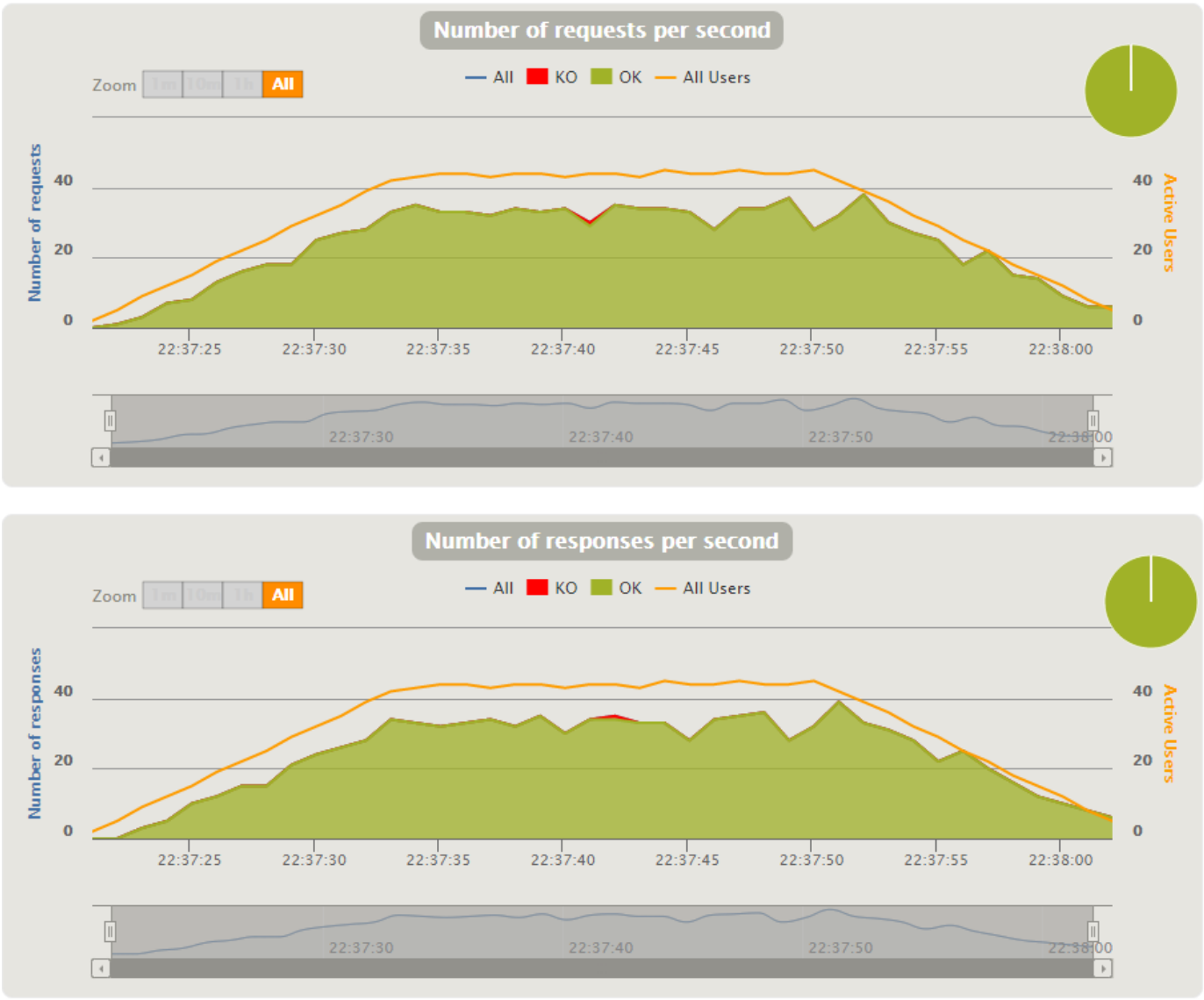


Figura 6.13: Escenario 3: Número de peticiones y respuestas por segundo

Se puede observar que la petición fallida se ha producido más o menos en la mitad de la simulación, no parece indicar que se haya producido por un error de sobrecarga, pues el pico de usuarios activos se produce unos segundos más adelante. Lo más probable es que dos peticiones hayan "colisionado" en un margen temporal muy reducido, por lo que se ha rechazado una de ellas,

aunque puede haberse producido un error interno del SOS (como por ejemplo, un acceso fallido a la base de datos).

Tiempo de respuesta del sistema

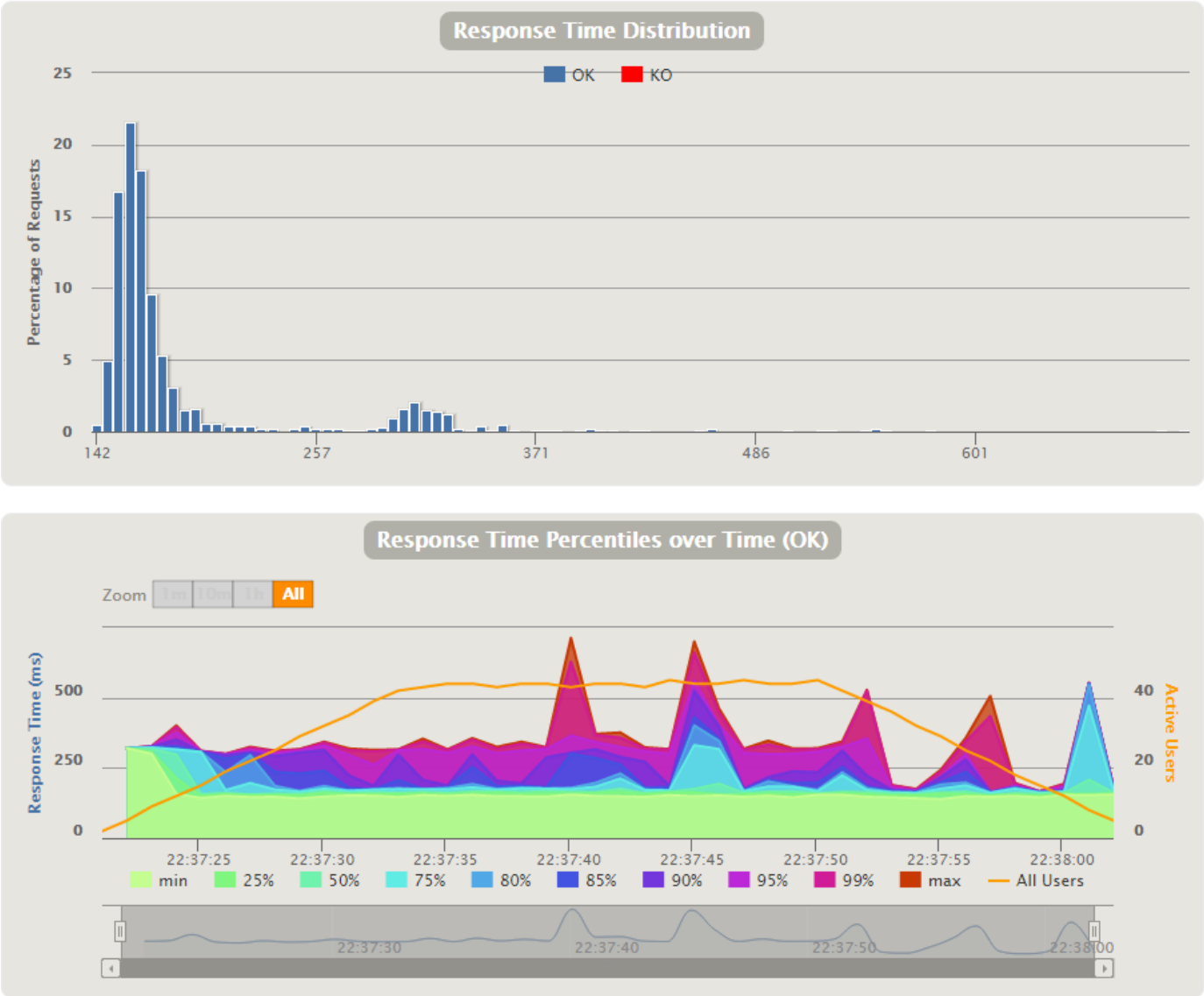


Figura 6.14: Escenario 3: Distribución y percentiles del tiempo de respuesta

En la primera gráfica se puede comprobar el valor que del percentil 95 que se ha comentado, pues la mayoría de las respuestas están agrupadas en torno

a los 140 y los 180 milisegundos. El valor de la respuesta fallida no aparece, pero se sitúa en los 428ms. En la segunda gráfica se muestra cómo el tiempo de respuesta ha producido dos picos durante la simulación, a partir de la mitad de la simulación (el primero pico coincide con la petición fallida), aunque durante prácticamente el resto de la simulación el tiempo de respuesta es menor de 300ms.

6.6. Escenario 4

6.6.1. Datos de entrada

En este escenario se introducirán 1000 usuarios durante 60 segundos (un usuario cada 0.06 segundos), cada uno enviando diez posiciones a intervalos regulares de un segundo. Podemos observar en la siguiente imagen la gráfica de usuarios activos resultante:

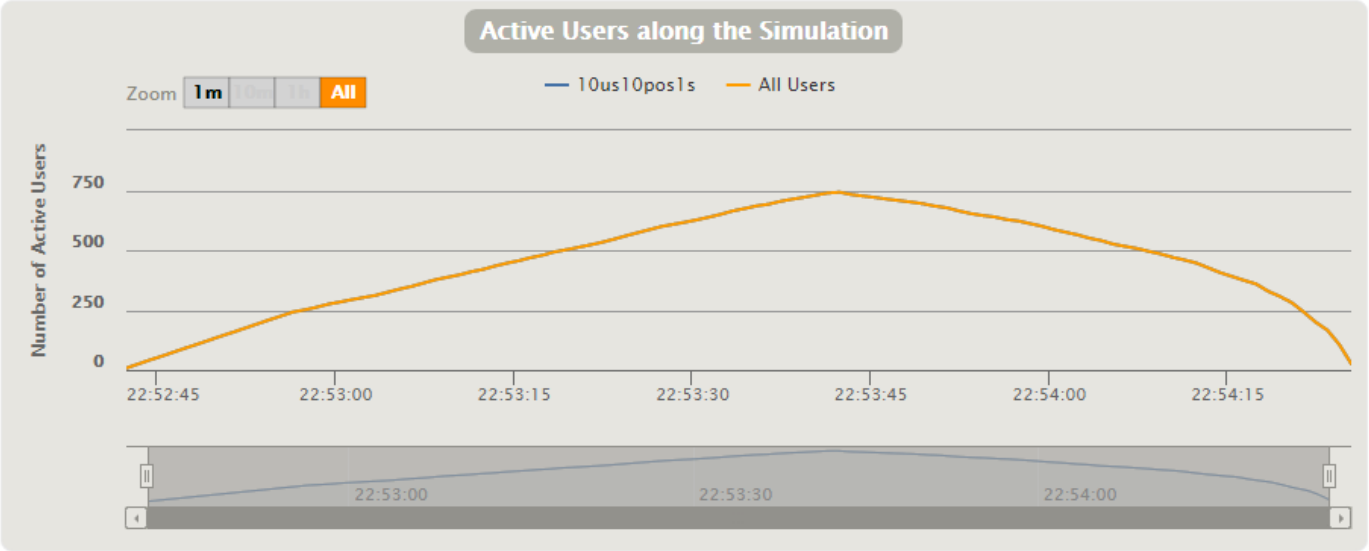


Figura 6.15: Escenario 4: Usuarios activos

La simulación dura casi 2 minutos, el pico de usuarios de forma simultánea son 742. En esta simulación podemos ver que la gráfica de usuarios activos cambia de forma con respecto a las simulaciones anteriores. Esto es debido

al tiempo de respuesta que veremos en los resultados, pues ha aumentado considerablemente y a dilatado en el tiempo el número de usuarios activos.

6.6.2. Resultados

Estadísticos

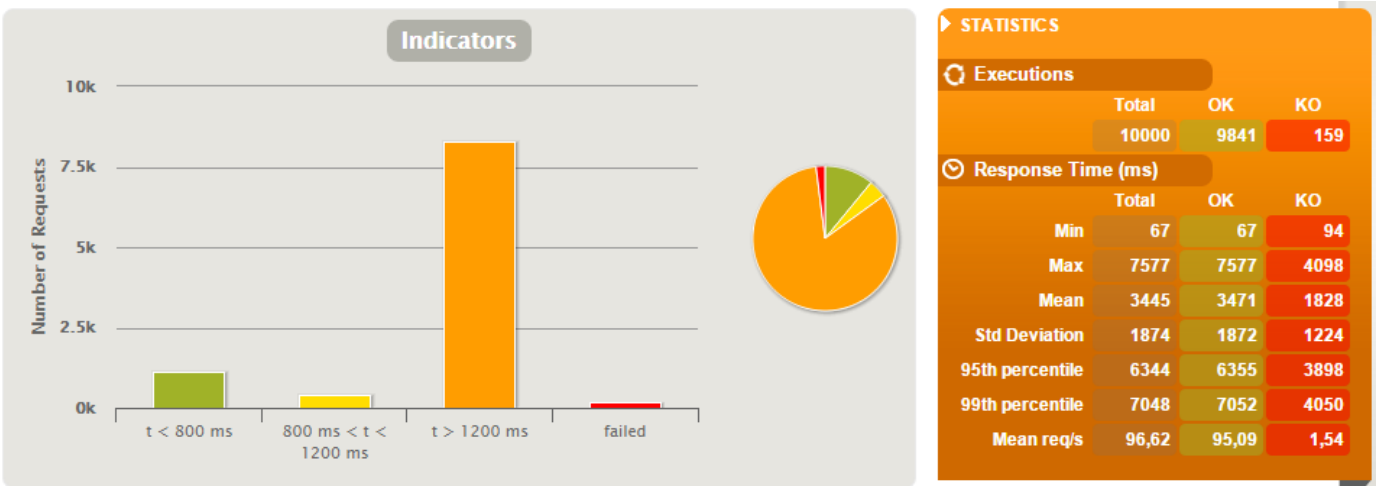


Figura 6.16: Escenario 4: Estadísticos

Se han realizado un total de 10000 peticiones, y de ellas han fallado 159. De las peticiones que se han realizado correctamente, la gran mayoría de ellas han tardado más de 1 segundo en ser respondidas, siendo el mínimo 67 milisegundos y el máximo 7.5 segundos. La media del tiempo de respuesta son 3.5 segundos, un valor justificado dada la cantidad de peticiones realizadas.

Número de peticiones y respuestas por segundo

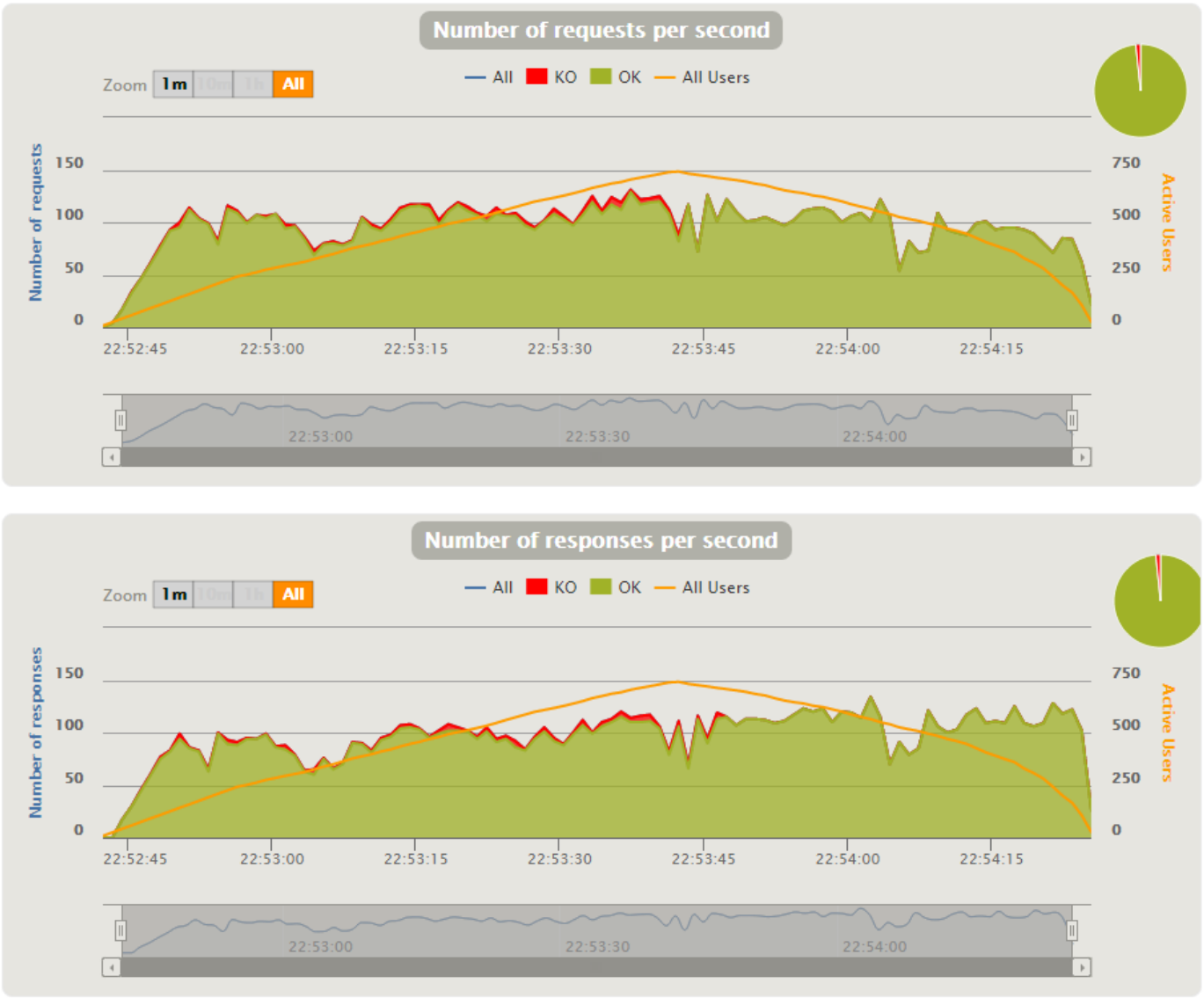


Figura 6.17: Escenario 4: Número de peticiones y respuestas por segundo

Se puede observar que el sistema ha aceptado peticiones y ha devuelto respuestas de una forma más o menos constante, lo que significa que distribuye bien el número creciente de conexiones, y dado el bajo número de peticiones fallidas en comparación, se puede deducir que el servidor intenta maximizar el número de conexiones activas a costa de aumentar el tiempo de respuesta.

Tiempo de respuesta del sistema

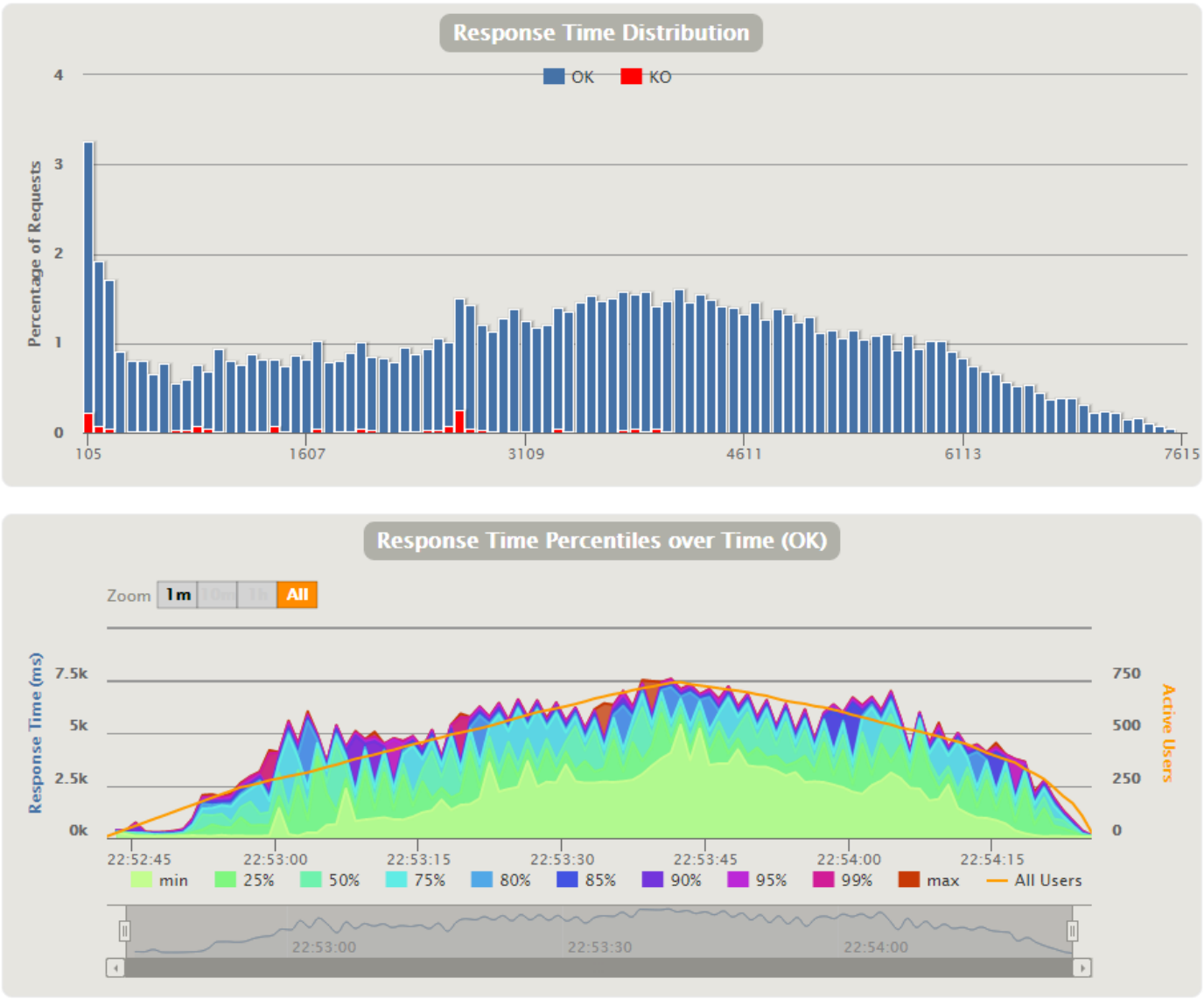


Figura 6.18: Escenario 4: Distribución y percentiles del tiempo de respuesta

Si observamos ambas gráficas de forma conjunta, podemos ver que el 9 % de las peticiones han sido respondidas con un tiempo de respuesta menor a medio segundo, y que éstas se han realizado al inicio y al final de la simulación. El resto de peticiones se distribuyen de forma más o menos regular entre 0.5 segundos y 6.5 segundos, y son menos frecuentes las peticiones que han

tardado más de 6.5 segundos.

En la gráfica inferior se puede ver cómo el tiempo de respuesta intenta seguir la forma de los usuarios activos, pero sin embargo no hay picos bruscos, ya que acepta conexiones de forma más lenta y regular en el tiempo, como hemos visto en la gráfica del número de peticiones y respuestas por segundo.

Capítulo 7

Conclusión y líneas futuras

En este capítulo vamos a concluir el proyecto realizado, primero analizando e interpretando los resultados obtenidos en el capítulo anterior. Seguidamente se presentarán una serie de mejoras y ampliaciones posibles que se pueden realizar a este sistema. Se terminará con una conclusión final de lo que se ha realizado en el proyecto.

7.1. Interpretación de los resultados

El objetivo de los diferentes escenarios de prueba que se han expuesto en el capítulo anterior, era realizar una prueba de carga al SOS, para observar la evolución en diferentes situaciones. A la hora de realizar las pruebas, se hizo un escenario más, con el objetivo de llegar a los 5000 usuarios activos de forma simultánea. Sin embargo, el servidor no pudo soportarlo y más del 90 % de las peticiones fallaron.

Igualmente, en el escenario 4 se puede observar que en el punto de carga máxima (742 usuarios) el tiempo de respuesta ha sido de 7.5 segundos, y dependiendo del contexto en el que se aplique este sistema, puede ser un retardo inaceptable.

La última gráfica del último escenario es que nos da los datos más valiosos, ya que si establecemos que el tiempo máximo de respuesta que permitimos es de, por ejemplo, 3 segundos, la cantidad de usuarios simultáneos pueden ser 250 o menos.

Tres segundos como tiempo de respuesta es prácticamente aceptable en cualquier situación que se necesite aplicar GEO-RBAC como forma de control

de acceso, por tanto, el procesador de eventos no necesita pedir el listado de sensores y sus posiciones en intervalos de tiempo menores a 3 segundos. Añadiendo un margen de error, el valor óptimo serían 5 segundos.

Por tanto, podemos concluir con que el sistema es capaz de funcionar de forma óptima para una cantidad máxima de usuarios simultáneos de 250 personas enviando posiciones cada segundo, con un margen de error de 5 segundos en la captura de posiciones. Si relajamos la condición de enviar posiciones cada segundo, el número de usuarios simultáneos aumentará.

7.2. Mejoras y extensiones posibles

Como ya se ha comentado, la finalidad de este proyecto era, entre otras, establecer una base para que en un futuro se pueda ampliar. De entre las posibles mejoras y extensiones del sistema en conjunto se pueden destacar:

- **Tener en cuenta la altura:** Agregar al usuario un nuevo atributo, la altura a nivel del mar, permitiría realizar GEO-RBAC dentro de edificios teniendo en cuenta los diferentes pisos que hay. Y poder restringir plantas o habitaciones en lugar del edificio completo.
- **SpatioTemporal-RBAC:** Se puede extender GEO-RBAC para que tenga en cuenta el tiempo en el que se produce el evento, de esta forma, se pueden reconocer patrones atípicos, por ejemplo: una persona entra todos los días a las 8 de la mañana y sale a las 5 de la tarde a un edificio que tiene permitido el acceso, pero si un día entra a las 10 de la noche, se le restringe el acceso. Configurado correctamente, y dentro de un margen de error, este sistema es muy potente a la hora de detectar intrusiones y accesos no autorizados.
- **PROX-RBAC y Roles dinámicos:** Otra de las posibles mejoras que se podría aplicar en muchas situaciones es tener en cuenta la asociación de posibles elementos con roles distintos, por ejemplo: un conductor de un camión no tiene acceso a una zona restringida, pero si se encuentra dentro de cierto tipo de camión (detectado por proximidad) se le puede otorgar permiso, asignando un rol especial a esa asociación.

En cuanto a la aplicación que se ha desarrollado, se podrían mejorar los siguientes aspectos:

- **Soporte para múltiples roles:** En esta fase del desarrollo, un dispositivo sólo podía tener un único rol, no sería excesivamente complicado

añadir soporte para múltiples roles, y que se pueda escoger cómo actuar en caso de haber una colisión (un dispositivo posee dos roles, uno tiene permiso para acceder a una zona, pero el otro no).

- **Soporte de más tipos de bases de datos:** Añadir en el archivo de configuración la opción de poder añadir más tipos de bases de datos, escribiendo en archivos distintos las sentencias SQL específicas de cada sistema de gestión de bases de datos.

- **Cargar de forma dinámica las entradas/salidas:** Mediante el uso de interfaces y genéricos de Java, se puede hacer que la aplicación cargue librerías que obtengan la entrada de las posiciones o que recojan los eventos de salida de forma dinámica, sin tener que recompilar la aplicación principal. Incluso se puede cargar en tiempo de ejecución mediante el objeto "Class Loader" de Java.

- **Traspasar la aplicación a un framework:** Existen frameworks que permiten integrar diferentes sistemas y realizar aplicaciones que permitan ser mucho más escalables, como por ejemplo, Spring sobre Hadoop.

7.3. Conclusión

Tal y como acabamos de explicar, el sistema y la aplicación pueden ser mejorados, sin embargo, desarrollando una aplicación que estaba más orientada a ser una "Demo" podemos ver que es funcional, y podría ser aplicado en una pequeña empresa sin problemas.

Podemos concluir asegurando que los objetivos principales que se han propuesto al inicio del proyecto han sido cumplidos satisfactoriamente:

- **Desarrollar un procesador de eventos GEO-RBAC integrable en cualquier sistema:** Aunque se ha aplicado a un caso de uso específico, en la implementación de la aplicación se puede ver cómo se pueden utilizar diferentes entradas (de hecho, hemos utilizado un simulador durante la fase de desarrollo, y se ha utilizado un SOS real en las pruebas) y también diferentes salidas que comparten un interfaz común.

- **Integrar el procesador de eventos GEO-RBAC en un sistema de sensores:** En las pruebas se ha demostrado que el sistema funciona correctamente.

- **Realizar pruebas de escalabilidad e interpretar los resultados:** En el capítulo 7 se han realizado las pruebas de escalabilidad necesarias para poder estudiar la ejecución óptima del sistema, que se ha expuesto en la sección anterior.

Anexo

Anexo A

Archivos de configuración

georbac.properties

```
server_ssl_port: 4433
server_port: 8088
map_active: example_teleco.json
sos_url: http://eogorriti.com:8080/52n-sos-webapp/service
report_interval: 500
default_role: default
```

login.properties

```
iglaub = iglaub, client
alcaptar = alcaptar, client
enolgor = enolgor, admin, client
```

Mapa utilizado en las pruebas (2 zonas, 2 roles):

```
 {
"type": "map",
"url":
    "http://maps.googleapis.com/maps/api/staticmap?center=39.479533,-0.342569&zoom=19&
"zones": [
  {
    "name": "teleco1",
    "allowed_roles": [
      "alumno",
      "profesor"
    ],
```

```
    "bounds": [
      "39.47978148672999,-0.34291505813599715",
      "39.479499930755146,-0.3430545330047615",
      "39.47926806027301,-0.3422284126281852",
      "39.47954961718647,-0.34208893775938737"
    ],
    "relative_bounds": [
      "190,196",
      "139,332",
      "445,449",
      "500,309"
    ]
  },
  {
    "name": "teleco2",
    "allowed_roles": [
      "alumno",
      "profesor"
    ],
    "bounds": [
      "39.48019553756376,-0.3432154655456461",
      "39.48012928959603,-0.34324765205384306",
      "39.480145851593896,-0.3432691097259632",
      "39.48006304156515,-0.3433012962341267",
      "39.48006304156515,-0.3433334827423237",
      "39.48004647954756,-0.3433549404144439",
      "39.48002991752606,-0.3433227539062469",
      "39.47996366940055,-0.3433549404144439",
      "39.47984773502903,-0.3429579734802375",
      "39.48002991752606,-0.3428721427917568",
      "39.480079603578794,-0.343086719512925",
      "39.480145851593896,-0.3430652618408049"
    ],
    "relative_bounds": [
      "78,2",
      "66,28",
      "57,26",
      "45,64",
      "34,61",
      "27,70",
      "39,80",
      "26,115",
      "175,168",
```

```
        "206,78",
        "128,55",
        "136,23"
      ]
    }
  ],
  "width": 640,
  "height": 640
}
```

Anexo B

Sentencias SQL y cel CEP

B.1. Sentencias SQL

Crear tabla:

```
CREATE TABLE IF NOT EXISTS device_roles (device TEXT NOT NULL,
   role TEXT NOT NULL);
```

Borrar tabla:

```
DROP TABLE IF EXISTS device_roles;
```

Seleccionar dispositivo:

```
SELECT device, role FROM device_roles;
```

B.2. Sentencias del CEP

Generar evento cuando un dispositivo entra a una zona:

```
select zone,device from PositionEvent where zone.contains(device)
    and not zone.isAllowed(device) and not
    device.getStandingForbiddenZones().contains(zone)
```

Generar evento cuando un dispositivo entra a una zona:

```
select zone,device from PositionEvent where
    device.getStandingForbiddenZones().contains(zone) and not
    zone.contains(device)
```

Anexo C

Mensajes intercambiados con los servicios

C.1. SOS

Petición para listar sensores:

```
   <?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3.org/2003/05/soap-envelope
        http://www.w3.org/2003/05/soap-envelope/soap-envelope.xsd">
    <env:Body>
        <sos:GetCapabilities
            xmlns:sos="http://www.opengis.net/sos/2.0"
            xmlns:ows="http://www.opengis.net/ows/1.1" service="SOS"
                xsi:schemaLocation="http://www.opengis.net/sos/2.0
                http://schemas.opengis.net/sos/2.0/sosGetCapabilities.xsd">
            <ows:AcceptVersions>
                <ows:Version>2.0.0</ows:Version>
            </ows:AcceptVersions>
            <ows:Sections>
                <ows:Section>OperationsMetadata</ows:Section>
            </ows:Sections>
        </sos:GetCapabilities>
    </env:Body>
</env:Envelope>
```

Petición para obtener la última posición:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3.org/2003/05/soap-envelope
        http://www.w3.org/2003/05/soap-envelope/soap-envelope.xsd">
    <env:Body>
        <sos:GetObservation
            xmlns:sos="http://www.opengis.net/sos/2.0"
            xmlns:fes="http://www.opengis.net/fes/2.0"
            xmlns:gml="http://www.opengis.net/gml/3.2"
            xmlns:swe="http://www.opengis.net/swe/2.0"
            xmlns:xlink="http://www.w3.org/1999/xlink"
            xmlns:swes="http://www.opengis.net/swes/2.0"
                service="SOS" version="2.0.0"
                xsi:schemaLocation="http://www.opengis.net/sos/2.0
                http://schemas.opengis.net/sos/2.0/sos.xsd">
            <sos:offering>offering/id_sensor</sos:offering>
            <sos:temporalFilter>
              <fes:TEquals>
                <fes:ValueReference>phenomenonTime</fes:ValueReference>
                <gml:TimeInstant gml:id="ti_1">
                  <gml:timePosition>latest</gml:timePosition>
                </gml:TimeInstant>
              </fes:TEquals>
          </sos:temporalFilter>
          <sos:responseFormat>http://www.opengis.net/om/2.0</sos:responseFormat>
        </sos:GetObservation>
    </env:Body>
</env:Envelope>
```

C.2. GCM

Informar de entrada/salida a zona no autorizada:

```
{
   "registration_ids": [array_id_dispositivos],
   "time_to_live": tiempo_de_vida_en_segundos,
   "id": "id_dispositivo",
   "data": {
      "zone": "nombre_zona",
      "type": "enter" //o "exit"
   }
}
```

C.3. HMI

Evento de entrada/salida a zona no autorizada:

```
{
   "type": "event",
   "event_type": "entered_forbidden_zone", // o
       "exited_forbidden_zone"
   "id": "id_dispositivo",
   "role": "rol_usuario",
   "position": "latitud,longitud",
   "relative_position": "x,y",
   "zone": "nombre_zona",
   "timestamp": marca_en_milisegundos
}
```

Bibliografía

[1] Open Geospatial Consortium Lance McKee, Senior Staff Writer. OGC History. http://www.opengeospatial.org/ogc/historylong.

[2] Open Geospatial Consortium Standards. http://live.osgeo.org/en/standards/standards.html.

[3] Sensor Web Enablement DWG. http://www.opengeospatial.org/projects/groups/sensorwebdwg.

[4] Arne Bröring, Johannes Echterhoff, Simon Jirka, Ingo Simonis, Thomas Everding, Christoph Stasch, Steve Liang, and Rob Lemmens. New generation sensor web enablement. *Sensors*, 11(3):2652–2699, 2011.

[5] Projects implementing SWE. http://www.ogcnetwork.net/SWE_Projects.

[6] Elisa Bertino. Rbac models - concets and trends. pages 511–514.

[7] David F. Ferraiolo and D. Richard Kuhn. Role-based access controls. *15th National Computer Security Conference*, pages 554–563, 1992.

[8] Salvatore Piccione Sergio Gusmeroli and Domenico Rotondi. A capability-based security approach to manage access control in the internet of things. *Mathematical and Computer Modelling*, 58:1189–1205, 2013.

[9] Michael S. Kirkpatrick, Maria Luisa Damiani, and Elisa Bertino. Prox-rbac: A proximity-based spatially aware rbac. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 339–348, New York, NY, USA, 2011. ACM.

[10] Subhendu Aich, Shamik Sural, and A. K. Majumdar. Starbac: Spatio-temporal role based access control. In *Proceedings of the 2007 OTM Confederated International Conference on On the Move to Meaningful Inter-*

net Systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part II, OTM'07, pages 1567–1582, Berlin, Heidelberg, 2007. Springer-Verlag.

[11] Gabriel Ghinita Michael S. Kirkpatrick and Elisa Bertino. Privacy-preserving enforcement of spatially aware rbac. *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING*, 9:627–640, 2012.

[12] Nimalaprakasan Skandhakumar, Jason F. Reid, Ed Dawson, Robin Drogemuller, and Farzad Salim. An authorization framework using building information models. *The Computer Journal*, 55(10):1244 –1264, October 2012.

[13] David Flanagan. *Java in a nutshell*. O'Reilly Media, 2005.

[14] Aboute SQLite. `http://www.sqlite.org/about.html`.

[15] Extensible Markup Language (XML). `http://www.w3.org/XML/`.

[16] Guía breve: XML. `http://www.w3c.es/Divulgacion/GuiasBreves/TecnologiasXML`.

[17] XML. `http://en.wikipedia.org/wiki/XML`.

[18] WSDL. `http://en.wikipedia.org/wiki/Web_Services_Description_Language`.

[19] JSON. `http://json.org/`.

[20] Introducción a los WebSockets: incorporación de sockets a la Web. `http://www.html5rocks.com/es/tutorials/websockets/basics/`.

[21] Javascript. `http://es.wikipedia.org/wiki/JavaScript`.

[22] What is Scala? `http://www.scala-lang.org/what-is-scala.html`.

[23] SOS sensors and data model. `http://www.ogcnetwork.net/sos_2_0/tutorial/om`.

[24] Sensor Observation Service. `http://52north.org/communities/sensorweb/sos/index.html`.

[25] Google Cloud Messaging Overview. `https://developer.android.com/google/gcm/gcm.html`.

[26] Implementing GCM server. `https://developer.android.com/google/gcm/server.html`.

[27] About the Eclipse Foundation. https://eclipse.org/org/.

[28] Maven. http://es.wikipedia.org/wiki/Maven.

[29] Apache Subversion. http://en.wikipedia.org/wiki/Apache_Subversion.

[30] ABOUT THE JETTY PROJECT. http://eclipse.org/jetty/about.php.

[31] Jetty (web server). http://en.wikipedia.org/wiki/Jetty_%28web_server%29.

[32] SOS 4.x Documentation. https://wiki.52north.org/bin/view/SensorWeb/SensorObservationServiceIVDocumentation.

[33] Gatling documentation. http://gatling.io/docs/2.0.3/.

Índice de figuras