Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

S4C: Services for Citizens. Desarrollo de servicios autónomos para los ciudadanos en el ámbito de las ciudades inteligentes

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Autor: Vicente Izquierdo García

Tutores: Vicente Pelechano Ferragud

Joan Fons i Cors

Curso 2015-2016

Resumen

En este proyecto se presenta un simulador de tráfico extensible y personalizable desarrollado siguiendo una arquitectura o plataforma IoT, en el que se trata de explorar servicios en el ámbito de las futuras ciudades inteligentes. Su valor principal es el de tener la capacidad de poder introducir a los ciudadanos (con diferentes perfiles) como un componente destacado dentro del sistema de manera no intrusiva gracias a una interfaz integrada en sus dispositivos móviles. La finalidad es proporcionar un sistema inteligente simulador de tráfico donde los componentes sean tratados como *smart objects*, los cuales auto-gestionan (de manera autónoma) su comportamiento, e interactúan entre sí. Por otra parte, los usuarios del sistema, los propios ciudadanos, puedan interactuar con el sistema gracias a una interfaz e influir en el comportamiento del mismo.

El sistema permite la definición y simulación de escenarios personalizados de tráfico compuestos de múltiples participantes de tráfico, tales como vehículos, semáforos, señales de tráfico, tramos de carretera e incidentes, pudiendo personalizar los atributos de cada una de sus instancias. Además, se puede desplegar en dicho escenario diferentes sondas de monitorización que avisen de cualquier evento de tráfico especificado y que permite a diferentes roles de usuarios a acceder en tiempo real a esta información, pudiendo incluso proporcionar feedback a las notificaciones del sistema en cualquier momento.

El proyecto desarrollado consta de: una plataforma IoT donde se despliegan los *smart objects* relacionados con el tráfico (vehículos, carreteras, señales, etc.), un simulador que permite 'animar' estos *smart objects*, y un servicio de interacción con los ciudadanos que hace uso de una aplicación móvil para permitir la interacción de estos *smart objects* autónomos con los usuarios. Se han utilizado para ello diferentes tecnologías y *frameworks* tales como Java, OSGi y Android para el desarrollo funcional del proyecto y MQTT y JSON para la mensajería y las comunicaciones asíncronas.

Palabras clave: Ciudades Inteligentes, Internet de las Cosas, Servicios en la Nube, Sistemas Auto-Adaptativos, Interacción Hombre Máquina

Abstract

The project's aims are to present a modular traffic simulator for IoT architectures and exploring those services in the scope of future Smart Cities. Its main value is to have the ability to introduce citizens (with different profiles) as an important component in the system through an integrated interface on their mobile devices. The purpose is to provide a smart traffic simulator system where components are smart objects. Those objects have a self-management (autonomously) behavior, and are capable of interacting with each other. Moreover, system users (citizens) are an influent component being able to interact with the system itself through an interface.

The system is capable of defining and simulating personalized traffic scenarios with different components like vehicles, traffic lights, road segments and incidents. All of them being able to customize. Moreover, the system can be monitored to notify different events to individual users and user roles. Those users can receive real time notifications from the system and provide feedback.

The project consists of: An IoT services platform where smart traffic objects (vehicle, traffic signals, roads...) are deployed in a simulated scenario, a simulator that allows to "animate" those smart objects and a user integration module with a mobile client to integrate users with the rest of the system. Different technologies and frameworks have been used for that propose. The most relevant are Java, OSGi and Android for the system development and MQTT and JSON for asynchronous communications.

Keywords: Intelligent cities, Internet of Things, Cloud Services, Auto-Adaptive systems, Men-Machine interaction.

Tabla de contenidos

Tabla de figuras

1. Introducción

1.1. Introducción

Las ciudades inteligentes del futuro están siendo construidas como ecosistemas de servicios altamente conectados entre sí con la participación activa de una gran cantidad de variedades de objetos físicos (edificios, vehículos, empresas, calles, semáforos, señalización de tráfico, redes eléctricas, etc.) teniendo como objetivo el poder proporcionar diferentes servicios avanzados e inteligentes a los ciudadanos. Estos sistemas se están materializando a través de plataformas como la Internet de las Cosas (IoT) o los sistemas ciber-físicos (CPS). En este ámbito será necesario construir servicios y plataformas que sean capaces de auto-gestionarse para que se puedan adaptar dependiendo de la situación en la que se encuentren estos sistemas, permitiendo proporcionar de cierta autonomía a la gestión de la ciudad.

Sin embargo, uno de los grandes retos presentes en este tipo de proyectos, es cómo gestionar la participación humana en estos procesos autónomos: desde informar al usuario del estado del sistema y notificarle cuando tome alguna decisión, para que éste sea consciente de la situación de estos sistemas, hasta involucrar al mismo en la toma de decisiones importantes en caso de conflictos o problemas que pusieran en riesgo la autonomía e integridad de los servicios y de los propios usuarios o simplemente para personalizarlo a los gustos y necesidades de los ciudadanos.

En este proyecto se propone explorar este tipo de soluciones a través del desarrollo de una plataforma de servicios auto adaptativos orientados a la gestión del tráfico en el ámbito de una ciudad inteligente tomando como referencia la plataforma VLCi (Valencia Inteligente) [1], construida sobre la plataforma IoT FiWare [2].

La solución desarrollada ha sido diseñada para poder ser construida usando los servicios de dicha plataforma, extendiendo los servicios que ya ofrece a la ciudad (información de tráfico, monitorización ambiental, logística, etc.) e incluye las características principales de computación autónoma involucrando activamente al humano (Human in the Loop [3][4][5]). El prototipo propuesto incluye además de una interfaz de usuario (en forma de aplicación móvil) que le permite al ciudadano obtener información en tiempo real del sistema e interactuar con él.

Es importante recalcar que, cada componente mantiene su propio contexto y se comunica de manera autónoma para informar de algún cambio en él. Así los semáforos, por ejemplo, informarán de su estado a las carreteras y éstas comunicarán su estado a su vez a los vehículos, que analizándolo determinarán si pueden avanzar al siguiente punto de su ruta, deben detenerse o deben cambiar su velocidad, entre otras acciones. En todo momento el sistema involucra activamente al usuario. Además, dicho sistema tiene la capacidad de comunicarse con los usuarios y notificarles, mediante un cliente móvil independiente del sistema, de diferentes eventos de tráfico que les puedan ayudar. Pudiendo el usuario responder a ciertos mensajes proporcionando información adicional que el sistema puede analizar y tomar en consecuencia ciertas decisiones en función de las decisiones del conjunto de usuarios, introduciendo al humando dentro del bucle de ejecución del sistema. Este modelo de interacción es denominado "*Human in the loop*".

1.2. Motivación del proyecto

El trabajo desarrollado en el presente proyecto está dirigido y enfocado principalmente al desarrollo de una infraestructura software IoT desplegable en plataformas reales con un enfoque eminentemente práctico. Por ello se tiene muy presente a la hora del diseño, de asegurar la compatibilidad de los servicios desarrollados en diferentes escenarios y el coste que supone cualquier cambio en las tecnologías empleadas.

Las motivaciones que han llevado a la realización del presente proyecto son las siguientes:

- **IoT y ciudades inteligentes**

La proliferación, cada día mayor, de dispositivos altamente conectados presentes en las ciudades que proporcionan una ingente variedad y cantidad de datos proporcionan un entorno novedoso para poder construir diferentes ecosistemas de servicios que utilicen dichos datos y beneficien tanto a ciudadanos como a empresas y administraciones.

- **Gestión inteligente de tráfico**

Las congestiones de tráfico son uno de los desafíos más críticos en la movilidad en áreas urbanas, lo que acaba provocando retardos en los desplazamientos, aumento del gasto de combustible, incremento de la emisión de CO2, y malestar general de los ciudadanos. Además, el reto de la selección de la ruta más óptima dependiendo del estado de las carreteras y las necesidades del entorno es muy importante para la reducción de la contaminación urbana y la densidad de tráfico. Por ello la necesidad de proyectos que introduzcan nuevos paradigmas para resolver este tipo de problemas.

- **Autonomía**

El número de servicios y de procesamiento de datos presentes en este tipo de proyectos puede ser enorme y la acción del ser humano puede reducir el rendimiento de estos sistemas debido a la necesidad de interacción. Es por ello la necesidad del diseño de sistemas que, sin necesidad de interacción, sean capaces de funcionar perfectamente y tengan la capacidad de analizar su entorno y tomar decisiones sin depender de ningún usuario. Pero sin excluirlo del todo, permitiéndole actuar en situaciones de peligro del sistema o para el aprendizaje del mismo.

- **Extensibilidad**

Existen algunos sistemas simuladores de tráfico en el mercado que realizan tareas similares a las propuestas en este proyecto, pero introducen demasiada complejidad y son muy rígidos en cuanto a modificaciones o ampliaciones. Además, suelen estar más enfocados al análisis de flujo del tráfico que en la simulación individual de cada componente. Con este proyecto se pretende crear un sistema más enfocado a los componentes individuales y a su auto-gestión, proporcionando una infraestructura con bajo coste de modificación y de ampliación de los servicios que proporciona.

- **Human in the Loop**

Los sistemas autónomos de las ciudades inteligentes tienen como objetivo final el servir al ciudadano y mejorar su vida en la ciudad, por lo que es comprensible en que estos sistemas de una manera u otra puedan comunicarse con ellos para informarles de cambios de estado que les afecten y además poder obtener su opinión para refinar su funcionamiento. Esta ha sido una de las motivaciones centrales del proyecto.

- **Seguridad y sostenibilidad**

Al incluir al usuario en el sistema, éste es capaz de informar sobre los eventos que van ocurriendo en él. Esta funcionalidad puede ser muy útil para desarrollar sistemas que incrementen la seguridad en carretera, pudiendo notificar en caso de accidentes de tráfico la localización a los operadores de tráfico y tomando las medidas preventivas necesarias. Este funcionamiento también sería muy útil para del mantenimiento de la infraestructura, pudiendo avisar al gestor de tráfico al momento del mal funcionamiento de un componente.

1.3. Objetivos del proyecto

Los objetivos sintetizados de las motivaciones de este proyecto son los siguientes:

- Diseño y desarrollo de una infraestructura autónoma de servicios inteligentes orientados a la gestión de tráfico utilizando plataformas IoT como fuente de información.

- Proporcionar de los mecanismos necesarios para que el usuario participe de manera activa a la hora de la toma de decisiones del sistema autónomo.

- Proporcionar las herramientas necesarias para el desarrollo de escenarios personalizados e implementar un prototipo para demostrar la factibilidad de llevar a cabo esta solución

1.4. Alcance del proyecto

El presente proyecto forma parte de un conjunto de proyectos en el que se pretende desarrollar una plataforma estandarizada de un sistema autónomo, inteligente y auto-adaptativo de la gestión del tráfico real de una ciudad en el que sean los propios vehículos los que, comunicándose con su entorno (carreteras, señalización, incidentes), sean capaces de reaccionar a dicha información y tomar decisiones por sí mismos. Teniendo como parte indispensable en la toma de decisiones la entrada de información de los ciudadanos, estando éstos integrados en las comunicaciones del sistema. Por ejemplo, si una carretera presente en la ruta de un vehículo está colapsada, éste, podría, mediante una serie de políticas definidas y los parámetros del usuario, decidir si alterar su ruta o seguir su camino.

Ante dicho alcance global, el del presente proyecto se limita al modelado y desarrollo del sistema simulador, de un primer prototipo simulado y de los componentes del sistema de tráfico necesarios, incluyendo, el comportamiento de cada componente de tráfico, las comunicaciones entre ellos, la monitorización y la introducción del usuario como componente activo.

1.5. Ámbito de uso

Aun tratándose de un sistema autónomo, el usuario se ha tratado de manera prioritaria, teniendo la comunicación del sistema con éstos un papel fundamental. Para el prototipo se ha limitado el ámbito de uso del sistema a dos roles diferentes que representan a dos grandes colectivos de futuros usuarios.

Estos se han denominado, **“Conductor de un vehículo”** y **“Gestor de tráfico”**. La interacción del sistema con dichos roles tiene como finalidad el notificar de diferentes eventos que puedan afectar a dichos usuarios y en el caso de los conductores poder enviar *feedback* al sistema para proporcionarle información adicional con la que tomar futuras acciones.

Independientemente de dichas comunicaciones, el sistema es autónomo, esto significa que puede funcionar sin ninguna interacción con el usuario, pero este siempre puede proporcionarle información adicional que pueda alterarle su estado.

1.6. Metodología empleada para el desarrollo del proyecto

Desde el principio del proyecto se opta por una filosofía de trabajo orientada a objetivos y muy cercana al pensamiento ágil. Esto permite dividir el proyecto en varias etapas bien diferenciadas y trabajar en cada una de ellas de manera iterativa hasta haber alcanzado el objetivo propuesto.

Se toman en cuenta las buenas prácticas de trabajo ágil descritas en el modelo ágil SCRUM [6]. Esto es debido a que para el presente proyecto es necesario la obtención de resultados de manera rápida y donde los requisitos son cambiantes o poco definidos. Además, dado a que el proyecto tiene un nivel alto de complejidad y de innovación, se ha necesitado pues un marco ágil y flexible.

En el siguiente diagrama de Gantt se puede ver gráficamente el plan de trabajo seguido, los *Sprints*, los plazos de entrega y las iteraciones seguidas de las siguientes tareas:

Nombres de las tareas
Sprint #1 Análisis del caso de estudio y toma de requisitos
1.1 Descripción del caso de uso
1.2 Toma de requisitos del proyecto
1.4 Definición de la arquitectura del proyecto
1.3 Definición de una primera versión del alcance
1.5 Definición de hitos y entregables
Sprint #2 Aprendizaje de las tecnologías propuestas
Sprint #3 Desarrollo de la infraestructura básica y simulación
3.1 Análisis de requisitos de cada componente
3.2 Diseño del componente del sistema
3.3 Implementación del componente e incorporación en el sistema
3.4 Prueba del funcionamiento de los componentes
3.5 Puesta en común y obtención de peticiones de cambios
3.6 Implementación de dichos cambios
Sprint #4 Desarrollo de la infraestructura de monitorización
4.1 Análisis de requisitos
4.2 Diseño de la infraestructura de monitorización
4.3 Implementación de las sondas de monitorización
4.4 Prueba funcional de las sondas
4.5 Puesta en común y obtención de peticiones de cambios
4.6 Implementación de dichos cambios
Sprint #5 Desarrollo de los componentes para el "Human in the Loop"
5.1 Análisis de requisitos del cliente de usuario
5.2 Diseño de la aplicación
5.3 Implementación Android del cliente de usuario
5.4 Prueba funcional de la aplicación
5.5 Puesta en común y obtención de peticiones de cambios
5.6 Implementación de dichos cambios

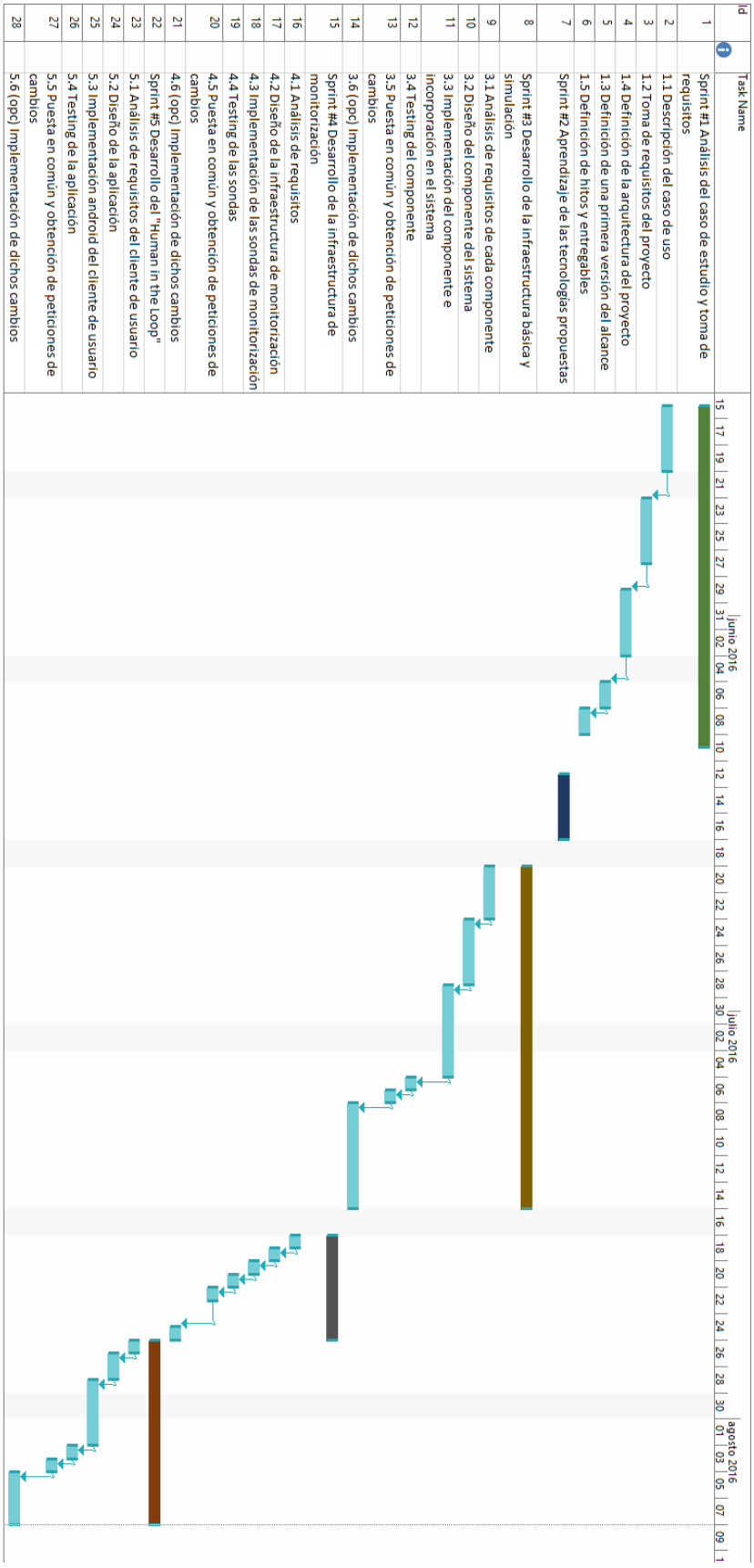

Figura 1: Diagrama de Gantt del proyecto

Como se puede observar en la figura anterior, el proyecto se ha dividido en 5 grandes hitos representados por los diferentes *Sprints* con un objetivo bien diferenciado en cada uno y que han tenido una duración acorde con la cantidad de tareas asociadas, dedicándole una media de 5 horas por día laboral. El equipo ha estado formado por 3 roles distintos; el "*Scrum master*" que supervisaba el proceso de desarrollo, el "*product owner*" que velaba por el correcto desarrollo del proyecto y el equipo desarrollador que realizaba el desarrollo del proyecto. Los dos primeros roles han sido desempeñados indistintamente por los tutores del proyecto y el equipo desarrollador, por el autor, contando con los tutores como expertos.

1.7. Organización de la memoria

La presente memoria se estructura en función de las diferentes fases de trabajo que se han seguido para desarrollar el proyecto en sí. Es decir, cada apartado principal corresponde con una etapa específica propia del desarrollo de software. Esto permite ver el proceso de decisiones tomadas a lo largo de todo el proceso y poder ver el alcance de éstas, desde que se piensan hasta que se plasman en la implementación. Como se ha comentado en el anterior apartado, para el desarrollo de la aplicación se ha seguido un modelo de desarrollo ágil. Sin embargo, para un mejor entendimiento de la memoria, se ha preferido explicarlo mediante un modelo en cascada.

Este modelo se compone de las siguientes etapas:

1. Especificación de requisitos
2. Diseño del software
3. Construcción o implementación del software
4. Integración
5. Pruebas
6. Despliegue
7. Mantenimiento

Basándonos en las etapas anteriores, la memoria está estructurada de la siguiente manera:

1. Este primer capítulo cuenta con la **introducción, motivación, objetivos y alcance** del proyecto.
2. El segundo capítulo, **describe el estado del arte y el contexto tecnológico** presente junto a proyectos similares y se explican las tecnologías utilizadas en el proyecto y el motivo de su elección.
3. El tercer capítulo, expone **el caso de estudio** donde se explica el contexto del proyecto y del escenario desarrollado sin llegar a entrar en detalles técnicos.
4. El cuarto capítulo, detalla **el análisis de los componentes** del proyecto. En él se expone la toma de requisitos y los atributos de los componentes y servicios.
5. El quinto capítulo, describe **la fase de diseño del sistema**. En él se detallan las decisiones tomadas de diseño, se describe la arquitectura del sistema, los patrones empleados, y se detallan los componentes del sistema, su funcionamiento y sus comunicaciones.
6. El sexto capítulo, detalla **la fase de implementación**, destacando la arquitectura empleada y la implementación de los aspectos más relevantes de cada servicio.
7. El séptimo capítulo, se evalúa el sistema bajo un **escenario** definido para demostrar su funcionamiento y se plantea como podría implementarse en un escenario real.
8. El octavo capítulo, comprende **las conclusiones** a las que se ha llegado durante el desarrollo del proyecto, así como los objetivos cumplidos y los aspectos más relevantes del desarrollo del mismo.
9. El noveno capítulo, se encuentran **las referencias** citadas en el presente documento.

2. Contexto tecnológico

2.1. Estado del arte

Durante los últimos años hemos podido ver como la red de redes (World Wide Web, WWW) se ha ido extendiendo a innumerables entornos de nuestra vida cotidiana tales como el internet móvil y la aparición de diferentes dispositivos "*Smart*" o inteligentes que, mediante diferentes sensores, su interconexión y el software adecuado, nos permiten el análisis inteligente de nuestro entorno.

La tendencia actual ante tal alto número de dispositivos inteligentes conectados a la red es la de la interconexión total de los mismos. Es lo que se ha denominado como "*Internet of Things*" o IoT. [7]

IoT hace referencia a un mundo totalmente interconectado, donde objetos y seres físicos interaccionan con entornos virtuales en el mismo espacio y tiempo. El objetivo es el de medir y controlar por completo nuestro entorno y poder planificar y reaccionar ante diferentes imprevistos pudiendo optimizar nuestras acciones.

El hecho de que internet esté presente al mismo tiempo en todas partes, permite que cualquier objeto sea perceptible de ser conectado y proporcione información comunicándose con otros objetos o con nosotros mismos. Esto podría ofrecernos infinidad de funcionalidades, pudiendo interactuar con ellos sin importar ni tiempo ni lugar.

Esto es posible mediante el despliegue de una estructura compleja, ya que no sólo requiere de sensores, si no que se necesita una estructura bien definida, una serie de estándares y protocolos, y lo que es más importante, software inteligente que utilice la información de dichos sensores e introduzca el componente "*Smart*" proporcionando valor a la información en bruto recogida del entorno.

Actualmente nos encontramos inmersos en una etapa en el que la industria aún está trabajando en la estandarización de las estructuras y estandarización de IoT y en el desarrollo de propuestas para diferentes sectores, como por ejemplo energía, industria, consumo o como en nuestro caso, transporte.

En este proyecto nos centramos en el ámbito de los transportes y en la gestión de recursos de una ciudad inteligente o "*Smart Cities*". Específicamente, se propone analizar el impacto y las oportunidades que este tipo de soluciones IoT podría ofrecer de cara a los ciudadanos, pudiendo hacerlos partícipes, de manera no intrusiva, en los procesos autónomos de la ciudad.

2.2. Proyectos similares y complementarios

La problemática del tráfico en ciudades y la necesidad de su control y optimización no es nada nuevo y por lo tanto existen actualmente numerosas aplicaciones y servicios que intentan proporcionar diferentes soluciones para ayudar a su gestión.

Actualmente existen soluciones para diferentes objetivos, sin embargo, utilizan un enfoque y presentan unas características que dificultaban su integración en infraestructuras orientadas a las ciudades inteligentes, motivo por el que se opta por el desarrollo de este proyecto.

A continuación, se exponen los tipos de soluciones existentes más relevantes y la finalidad que estos tienen:

- **Sistemas de información geográfica (GIS)**

  La finalidad es la de mostrar gráficamente sobre un mapeado grandes cantidades de datos actualizados provenientes del mundo real que están vinculados a una referencia espacial. En el caso de tráfico se representa información relacionada con el flujo y la densidad de tráfico de una zona concreta. Un ejemplo comercial de estos sistemas es **TransModeler** [8]. Este simulador ofrece una amplia gama de tareas centradas en la planificación de tareas y modelado del tráfico de grandes núcleos urbanos representándolos en un entorno 3D. Este tipo de simuladores están orientados al análisis del tráfico y a la predicción de modelos, pero su complejidad y al ser una versión comercial no permite la modificación ni ampliación del sistema.

- **Sistemas de análisis de flujo de tráfico**

  La finalidad de este tipo de proyectos es la de mostrar el comportamiento del flujo de tráfico en un mapa predefinido pudiendo modificar ciertos parámetros configurables. Un ejemplo de este tipo de aplicaciones es **Traffic-Simulation.de** [9] que nos permite observar de manera muy gráfica el comportamiento del flujo de vehículos en una serie de carreteras virtuales, permitiéndonos modificar ciertos parámetros, pudiendo ver su efecto en la simulación en directo.

- **Sistemas autónomos de transporte**

  Otros proyectos más similares a éste, introducen el concepto de la autonomía y la auto-adaptabilidad. Proyectos *Open Source* como **ADASIM** [10] permiten la simulación autónoma de problemas de enrutamiento de tráfico y permite el modelado de sus componentes (mapa, vehículos, tramos de carretera, etc.). Sin embargo, también tiene como objetivo el análisis del flujo de tráfico y no permite interactuar con los recursos como objetos inteligentes (*smart objects*).

Todas estas soluciones tienen como objetivo analizar el flujo de vehículos y ver su evolución. Son utilizados para hacer análisis de cambios y para comprobar el comportamiento sobre una infraestructura vial real, pero ninguno comparte nuestro objetivo de integrar al usuario al sistema y de obtener los datos actualizados mediante plataformas IoT.

Como se ha descrito en la motivación y en los objetivos del proyecto, el enfoque del presente proyecto no es el de desarrollar un sistema para la simulación del tráfico para observar el comportamiento del flujo de tráfico bajo ciertos condicionantes, ni el de optimizar los patrones de circulación de los vehículos.

El enfoque empleado está más orientado hacia proporcionar un sistema que pueda utilizar información de escenarios reales con una simulación autónoma de cada componente del sistema, donde cada componente se comunique independientemente, comparta su estado y donde el usuario (ciudadano) tenga un papel importante en las decisiones del sistema. Sin olvidar de que no sea rígido y complejo para que sea fácilmente extensible.

Para la toma de información de sistemas reales se ha tomado como plataforma de referencia, Valencia Inteligente (**VLCi**) [1]. Esta plataforma de servicios proporciona abiertamente al ciudadano de información permanentemente actualizada de más de 45 servicios municipales de la ciudad de Valencia, entre los que destacan áreas como la gestión de tráfico (la cual se emplea en el proyecto), sistemas de riego, seguridad ciudadana y medio ambiente. Este sistema utiliza la tecnología **FiWare** [2] que se encarga de proporcionar una API estandarizada para facilitar el desarrollo de aplicaciones inteligentes basadas en la utilización de múltiples sensores.

2.3. Tecnologías empleadas

Para el desarrollo del proyecto se utilizan diferentes tecnologías, lenguajes de programación, *frameworks* y modelos de comunicaciones orientados, en su mayoría, al desarrollo de proyectos IoT debido a sus características o a que se crearon como herramientas con ese propósito. A continuación, se describen brevemente los más importantes y el motivo de su elección.

2.3.1. OSGi

La naturaleza del sistema desarrollado está fundamentalmente orientada a proveer de diferentes servicios y librerías de manera independiente pero que de alguna manera puedan compartir información entre sus componentes para poder proporcionar un contexto compartido.

Open Services Gateway Initiative (OSGi) [11] [12] es una asociación de empresas creada para definir un estándar abierto para el desarrollo de pasarelas de servicios y también se refiere a un *framework* Java que nos permite definir una arquitectura para el desarrollo y el despliegue de aplicaciones y librerías modulares. OSGi define un modelo de desarrollo donde las aplicaciones, denominados *plug-ins* o *bundles*, son dinámicamente formados por diferentes componentes reutilizables dentro de un contexto compartido. Además, permite esconder las implementaciones internas de los componentes de los servicios, pudiendo abstraer unos de los otros facilitando la reutilización de los mismos y su comunicación.

El sistema autónomo ha sido desarrollado en el lenguaje de programación Java utilizando el *framework* OSGi. El *framework* ha permitido, como ya se ha mencionado, desarrollar los servicios de manera independiente bajo un mismo contexto, permitiéndonos cambiar su configuración dinámicamente en tiempo de ejecución y permitiendo la abstracción de dichos servicios. Esta abstracción permite cambiar la implementación interna de cualquier componente sin preocuparse de las dependencias con los demás. Esto nos facilita mucho el desarrollo de nuevos servicios o en ampliaciones de los existentes.

2.3.2. MQTT

El diseño e implementación de un sistema general de comunicaciones entre componentes tiene una gran importancia dentro del proyecto. Aun habiendo generado un escenario simulado, el proyecto ha sido desarrollado pensando en su despliegue en un escenario real, donde los componentes no se encuentran en el mismo entorno de ejecución y necesitan de alguna plataforma de comunicación externa para transmitir y recibir información de los demás. Además, es importante que dichas comunicaciones sean asíncronas y ligeras, ya que, no hay una consciencia por parte de los componentes de la existencia de todos los demás. Por ejemplo, una carretera no tiene por qué conocer todos los vehículos que la están transitando, ni tampoco existe un servidor que le proporcione dicha información. Las comunicaciones por lo tanto tienen que ser asíncronas, ligeras e indirectas. Por ello se opta por un modelo de comunicación con patrón subscriptor/publicador.

Dichos motivos son la clave para la elección de MQTT [13] [14] como *framework* principal de comunicaciones dentro del sistema. MQTT, o **MQ Telemetry Transport** es un protocolo de mensajería ligero de tipo publicador/subscriptor desarrollado por IBM diseñado expresamente para proyectos IoT donde las comunicaciones son entre dispositivos (Comunicaciones Machine To Machine, M2M) con bajo ancho de banda, bajo redes inseguras y donde la batería es un recurso crítico.

2.3.3. JSON

Para las comunicaciones M2M es necesario establecer un formato estándar para que los servicios pudieran entenderse y acceder a la información de los mensajes de manera estructurada. Para dicho formato se utiliza JSON. El motivo principal de su elección frente a XML como formato para la mensajería ha sido su mayor facilidad de desarrollo e integración de los analizadores y conversores de mensajes dentro de los servicios escritos en Java.

JavaScript Object Notation (JSON) [15] es un formato ligero de intercambio de mensajes fácil de leer y escribir por los humanos. Está basado en el lenguaje de programación JavaScript, pero es un lenguaje totalmente independiente del mismo pudiéndolo implementar en la mayoría de lenguajes. C, C++, C#, Java, Perl y Python entre otros.

2.3.4. Android (Java)

Para el diseño y desarrollo del cliente de usuario se opta por una aplicación nativa para dispositivos con el sistema operativo **Android** [16]. Los motivos principales para su elección son principalmente la reutilización de componentes y librerías del sistema, ya que son basados también en el lenguaje de programación Java. Además, la popularidad actual de dichos dispositivos, la movilidad que proporcionan y la facilidad de distribución de aplicaciones y su uso han motivado su elección.

Gracias a las funcionalidades de los sistemas Android, a sus guías de diseño y de modelización se consigue diseñar e implementar un cliente no intrusivo al dispositivo del usuario. Este cliente funciona en segundo plano sin consumir excesivos recursos y notifica asíncronamente al usuario de los eventos que le sean relevantes, pudiendo este además enviar *feedback* de forma intuitiva. El resultado se puede comprobar en el capítulo “7.1.3. Interacción con el usuario”.

3. Caso de estudio

3.1. Introducción del escenario “S4CSmartTraffic”

Actualmente vivimos en un mundo donde existen innumerables dispositivos interconectados que nos permiten obtener todo tipo de información a tiempo real. Siguiendo con esta tendencia, el desarrollo de servicios que aprovechan dichos datos y los procesan para proporcionar información útil a los usuarios está en pleno auge de demanda.

Dentro de estos servicios, el presente proyecto se enmarca dentro de los sistemas autónomos que, a partir de la información recogida del entorno, sean capaces por sí mismos y sin necesidad de la intervención humana, de tomar decisiones, cambiar su estado y adaptarse dinámicamente a su situación. En particular, se enmarca en el ámbito de las ciudades inteligentes, proyectos donde se pretende la construcción de ecosistemas de servicios altamente conectados con la participación activa de objetos físicos (edificios, vehículos, comercios, calles, semáforos, señales, etc.) para ofrecer servicios complejos a los ciudadanos. Estos sistemas se están materializando a través de plataformas como la Internet de las Cosas (IoT) o los sistemas ciberfísicos (CPS) [17].

En el presente proyecto se desarrolla la infraestructura del sistema y un prototipo para la gestión del tráfico de manera autónoma de una ciudad inteligente donde el usuario tiene un papel activo. Además, se pretende establecer una base para futuros proyectos que expandan las funcionalidades y la difusión de servicios para el ciudadano.

Se propone, como escenario base, una gestión del tráfico sensible al contexto y autónoma con un sistema de notificación y *feedback* por parte de los ciudadanos. La finalidad de este escenario es demostrar la gestión autónoma de los componentes de tráfico y las comunicaciones entre ellos y con los ciudadanos. En él, se puede demostrar cómo los vehículos gestionan su navegación dependiendo de las condiciones de su contexto (analizando la densidad de tráfico, incidentes de tráfico, semáforos en rojo etc.) de manera que influya en su control. En todo momento, el sistema autónomo mantendría informados a los operadores de tráfico sobre incidencias relevantes o problemáticas que requieran su atención, para que puedan tomar decisiones de actuación sobre el servicio.

También permitirá a los conductores de la ciudad, mediante una aplicación móvil, expresar su *feedback* sobre las notificaciones del tráfico recibidas, que a su vez serian tenidas en cuentas para valorar la utilidad de dichas notificaciones o adaptar, si fuera necesario, el comportamiento de algún componente. Este escenario actualmente estaría implementado en un simulador, pero este está diseñado expresamente para poder ser implementado en un sistema real, por ejemplo, en la plataforma de la ciudad de valencia, VLCi.

3.2. Dinámica del escenario

En el escenario propuesto se representa el sistema de tráfico de una Smart City donde todos los participantes tienen la capacidad de comunicarse con los demás mediante una serie de sensores y protocolos de comunicación. A continuación, se propone una descripción del mismo y del comportamiento esperado que se le ha querido dotar a la simulación del sistema mediante este proyecto.

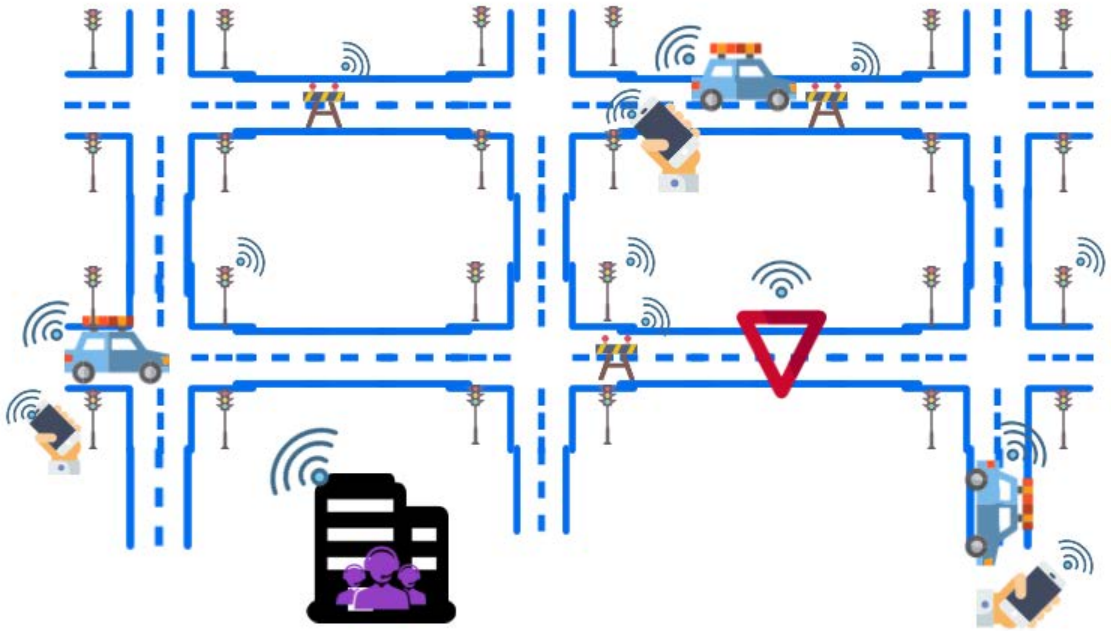

Figura 2: Diagrama general del escenario

En la anterior figura se pueden observar los diferentes elementos, o *Smart Objects*, que conforman el escenario. A continuación, se describe el comportamiento esperado del sistema y el papel que tiene cada componente en él.

El escenario representa a un conjunto de carreteras y elementos de tráfico de una hipotética futura ciudad inteligente donde todos los componentes de tráfico son inteligentes y capaces de comunicarse entre sí para establecer su comportamiento y actuar en función de su entorno de manera autónoma sin necesidad de intervención humana. En él, los vehículos son capaces de circular por los diferentes tramos de las carreteras y comunicarse directamente con ellas. Al entrar y salir de una carretera, el vehículo le comunicará de esta acción a la propia carretera para que sea consciente de ello y esta a su vez le comunicará el estado en que se encuentra en ese momento. Esto les permite indicar a qué velocidad máxima puede circular, si algún semáforo le impide continuar por su ruta o si existe una incidencia que le va a hacer detenerse. Los componentes de regulación del tráfico a su vez comunicarán su información a la carretera para que esta sea consciente y pueda determinar el estado que posteriormente envía a los vehículos que la recorren.

Independientemente de este funcionamiento autónomo, en caso de ocurrir un incidente en alguna carretera, la afectada emitirá un aviso de incidente a los gestores de tráfico al cargo para que éstos puedan actuar para resolverla. Además, para prevenir congestión de tráfico en las carreteras, éstas al detectar que el número de vehículos se acerca mucho a su capacidad máxima, emitirá un aviso a todos los conductores que tengan intención de atravesarla de que su estado está congestionado, pudiendo estos cambiar de ruta para evitar el atasco y permitiendo aligerar el tráfico por dicha carretera.

Todo este comportamiento global se delega en el comportamiento autónomo de cada componente inteligente consiguiendo un escenario autónomo donde los usuarios forman parte del mismo. A continuación, se describen con más detalle dichos componentes:

El vehículo, representa a cualquier tipo de vehículo que pueda circular por la ciudad (motocicleta, coche, camión, etc.). Su funcionalidad es la de moverse por el recorrido definido en el escenario para llegar de una posición origen a destino. Se espera para ello que sea capaz durante todo ese trayecto de comunicarse con los demás componentes de tráfico y respetar las leyes de circulación establecidas, por ejemplo, parándose ante un semáforo en rojo o regulando su velocidad cuando una señal de tráfico se lo indique. Para ello deben de seguir una **ruta** definida en su sistema **de navegación** que es el encargado de gestionar la ruta y dirige al vehículo hacia las siguientes posiciones del recorrido.

El sistema de navegación, representa al GPS del vehículo y es el encargado de gestionar la ruta del vehículo y de sugerir el movimiento del mismo (en caso de vehículo autónomo podría emplearse para gestionar también el movimiento para suplir la falta de conductor). Para ello, se espera de él que pueda controlar el movimiento y la posición del vehículo en todo momento pudiendo consultar en cada movimiento el estado de los componentes de tráfico que puedan afectar al movimiento del mismo. Por ejemplo, comprobar si hay un semáforo en rojo que impida moverse a la siguiente posición de la ruta.

El tramo de carretera, representa a cada uno de los tramos entre intersecciones de carreteras. Su finalidad es la de proveer a los vehículos de un camino donde poder moverse. El conjunto de tramos unidos por intersecciones forma el mapeado que los vehículos pueden recorrer para llegar de su origen a su destino, que a su vez serán posiciones determinadas en un tramo específico. Como *Smart Object*, se espera que cada tramo sea capaz de proporcionar su estado de manera universal por los demás componentes. Deben en todo momento poder proporcionar datos sobre sus atributos, como la velocidad máxima que permiten o del número de vehículos que la están circulando en el momento o si presentan incidencias y ser capaces de llevar un control de los vehículos que la recorren y ser consciente de si ocurre un incidente en ellas. Además, al llevar ese control del número de vehículos, al estar saturadas notificarán a los conductores afectados con dicha información.

El semáforo, tiene la funcionalidad de regular el tráfico en la posición donde se encuentra. Se espera de él que se pueda comunicar con el tramo de la carretera asociada para indicarle de sus cambios de estado, así esta podrá informar del estado de todos los semáforos asociados. Además, se caracteriza por tener un funcionamiento dinámico, pudiendo cambiar de estados cada cierto tiempo, para así no detener el tráfico permanentemente e individual, pudiendo tener cada semáforo un comportamiento distinto.

Las señales de tráfico, al igual que los semáforos, se encargan de la regulación del tráfico, pero influyen de diferente manera. Éstas pueden ser señales de velocidad máxima, de prohibido el paso, etc. Se espera de ellas que puedan comunicar su estado a las carreteras para que estas, al igual que con los semáforos, puedan transmitir su estado con la situación de todos los componentes de regulación de tráfico presentes en ella.

Incidentes de tráfico, representan a accidentes u otro tipo de eventos inesperado ocurrido en un tramo de carreteras. Su finalidad principal es la de avisar de que ha ocurrido un imprevisto y denegar el paso de vehículos por las zonas afectadas. Para ello se espera que sea capaz de comunicarse con el tramo afectado para que este tome las medidas establecidas, parando el tráfico y enviando una notificación de incidente a los agentes de tráfico.

Conductores y gestores de tráfico, representan a los ciudadanos del sistema y cada colectivo está interesado en diferentes notificaciones del sistema. Se espera que los conductores del vehículo reciban en su dispositivo móvil de notificaciones que le sean útil a lo largo de su ruta, por ejemplo, que cierto tramo tiene tráfico saturado, y en el caso de los gestores de tráfico, que reciban alertas en su panel de control de incidentes en algún tramo de carretera del mapa.

4. Análisis

4.1. Introducción

Uno de los objetivos principales del proyecto es el del **desarrollo de la infraestructura autónoma y de los servicios inteligentes** para poder simular el comportamiento del tráfico en un escenario virtual, siendo dichos componentes controlados por un simulador. En este capítulo se describe el proceso de análisis del escenario del que se extraen los componentes, las relaciones y las comunicaciones en el dominio de tráfico.

Para modelar el sistema de tráfico de un entorno real ha sido necesario diseñar objetos que representen una versión simplificada de su homologo, por tanto, ha sido necesario el diseño de objetos como las carreteras, los vehículos, las señales, los semáforos... Al tratarse de una primera versión se ha procurado generalizar y simplificar lo máximo posible estos objetos para centrarnos más en las funcionalidades que ofrecen más que en su similitud con su homólogo real.

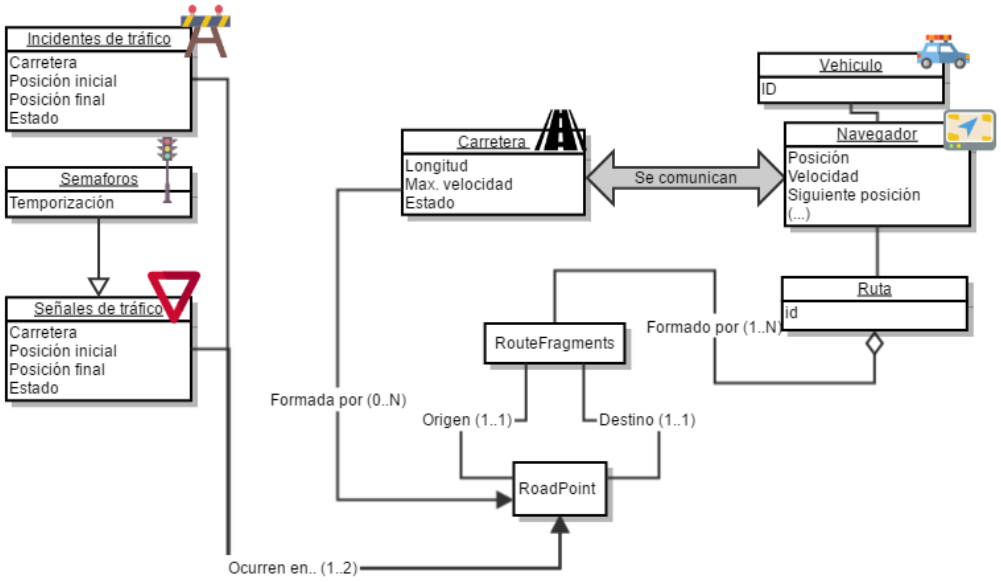

Figura 3: Visión general de los elementos del dominio de tráfico

En los siguientes capítulos se describe con mayor detalle las clases y sus relaciones presentes en la anterior figura. Este representa el resultado del análisis del escenario donde se puede ver una primera aproximación de los componentes más importantes del sistema.

4.2. Segmentos de carreteras

Los **segmentos de carreteras**, junto a los vehículos, son los elementos de tráfico centrales de la simulación. Se tratan de una representación simplificada de las carreteras del mundo físico, es decir, el diseño solo incluye una serie de atributos utilizados en el escenario planteado, pero la arquitectura permite la inserción de nuevos para futuros escenarios. Las intersecciones de varios segmentos por diferentes posiciones de los mismos crean un circuito el cual puede ser recorrido por el conjunto de vehículos.

Elementos de unión entre carreteras tales como rotondas y cruces múltiples han sido tratados por igual para simplificar el mapeado. Recalcar que, a cada carretera se le pueden atribuir diversos elementos de tráfico que obstaculicen y controlen la circulación de los vehículos en sus posiciones. Dichos elementos se describirán más adelante.

Los atributos **dinámicos** son aquellos que durante la ejecución del sistema suelen cambiar constantemente de valor. Suelen representar el estado del objeto o algún atributo que varíe en función del tiempo o en la ejecución. Como contrario se encontrarían los atributos **estáticos** presentes en la clase configurador del objeto que representarían los atributos que definen al objeto y no cambian a lo largo de la ejecución.

En las carreteras, en contraste con los vehículos, la mayoría de atributos son estáticos y por lo tanto están presentes en el configurador. Su único atributo dinámico es el estado.

- **Estado**: Indica si la carretera en un momento dado tiene tráfico fluido, denso, saturado o si la carretera se encuentra cerrada.

Los estáticos son los siguientes:

- **Longitud**: Indica la longitud de la carretera, o el número de posiciones que un vehículo tendrá que recorrer.
- **Capacidad**: Indica el número máximo de vehículos que la carretera puede soportar. Tiene relación con el estado.
- **Velocidad máxima**: Indica un máximo de velocidad por defecto para los vehículos. En caso de haber señales de tráfico que limiten la velocidad, esta prevalece.

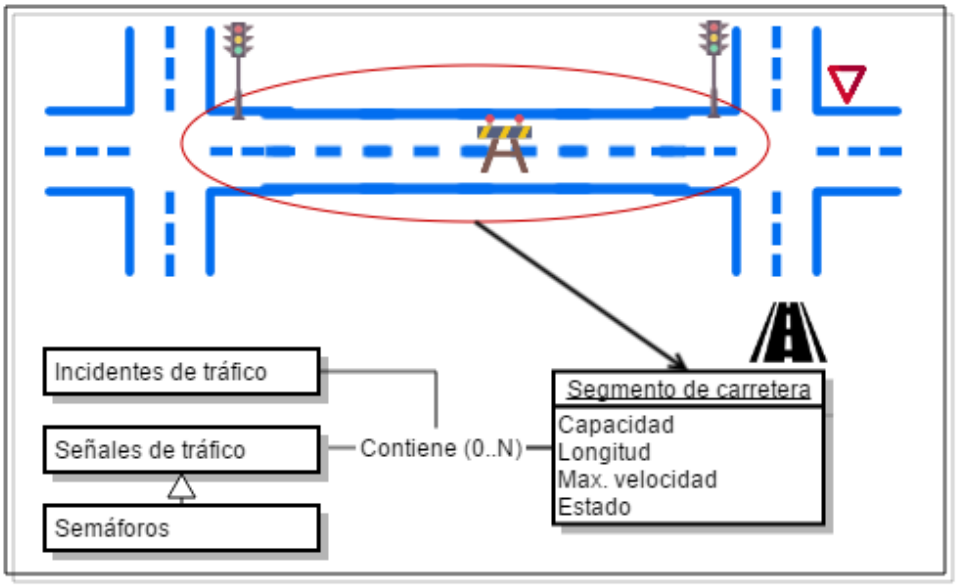

Figura 4: Diagrama de clase "Segmento de carretera"

4.3. Vehículos

Los vehículos tienen como objetivo el seguir una ruta formada por diferentes tramos de carreteras hasta llegar a su destino teniendo en cuenta atributos internos como la velocidad y atributos del contexto como el estado de las carreteras o la existencia de obstáculos que le impidan la circulación. El objetivo ha sido el de simular de una manera simplificada pero realista el funcionamiento real de la circulación de un vehículo.

El funcionamiento del vehículo es complejo y depende de muchas variables, tanto internas como recibidas del entorno por lo que ha sido necesario delegar la funcionalidad del mismo en varios submódulos relacionados que tienen fácil correspondencia con objetos del mundo real. Estos se describen en las siguientes secciones.

El vehículo se caracteriza por tener atributos altamente dinámicos, ya que se simula su movimiento a lo largo de la simulación y dicho movimiento depende en gran parte del entorno donde se mueve. Destacan los siguientes:

- **Velocidad**: Representa la velocidad del vehículo, o cuantas posiciones de su ruta avanzará por unidad de tiempo. Es dinámica ya que puede variar ante limitadores de velocidad como la velocidad máxima de una carretera o la presencia de señales de tráfico.
- **Estado**: Representa el estado de movimiento del vehículo, si está en movimiento, parado o estacionado.

Los atributos **estáticos** implícitos del vehículo son: el **identificador** del mismo y una referencia a su objeto **navegador**, el cual se describe a continuación.

4.4. Navegador

Una decisión importante en la definición del vehículo es la de separar los atributos y funcionalidades del mismo de las propias a su navegación por su recorrido. Esto último ha sido delegado a una clase dependiente del vehículo denominada **Navegador.** Funcionalidades como la definición y gestión de la ruta, el mantener la posición actual e indicarle al vehículo a qué posición tiene que avanzar en cada paso de simulación son delegadas en él. El navegador simbolizaría como si fuera el sistema de navegación o GPS de un vehículo real.

Para ello se han introducido los siguientes atributos, todos ellos con valores **dinámicos**.

- **Ruta del vehículo**: Representa una serie de tramos de carreteras que el vehículo tiene que atravesar para llegar a su destino, o posición final en la ruta.
- **Puntero en la ruta**: Representa el tramo de la ruta en la que se encuentra el vehículo en un momento dado.
- **Posición**: Indica la posición exacta donde se encuentra el vehículo en un momento dado.

La ruta del vehículo es considerada dinámica ya que podría ser modificada durante la ejecución, representando así un cambio por una ruta alternativa a la definida inicialmente.

Además, el navegador es el encargado en cada paso de simulación de comunicarse con el entorno y comprobar si el vehículo puede avanzar a la siguiente posición que le marca su ruta. Algunos eventos que le impedirían el paso podrían ser:

- Semáforo en rojo en una posición anterior a la posición de destino
- Carretera cerrada o colapsada en caso de intentar entrar en ella
- Incidente en una posición anterior a la posición de destino

Estos eventos están incluidos en el escenario planteado y hasta que no son resueltos no permiten la circulación del vehículo por la posición de la carretera donde están.

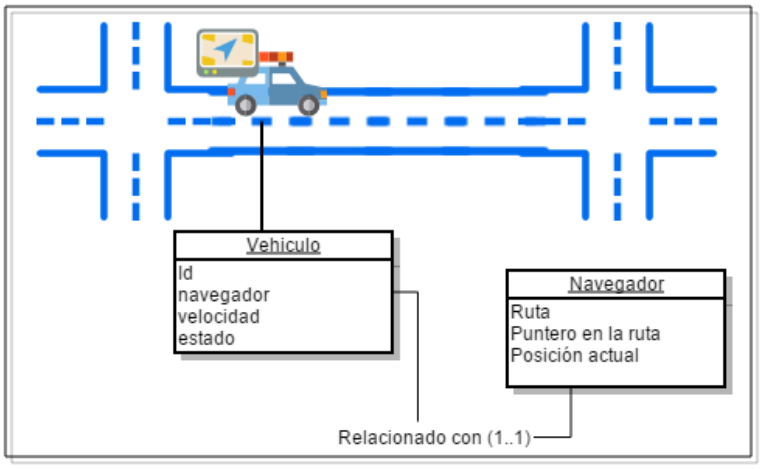

Figura 5: Diagrama de clases "Vehículo y Navegador"

4.5. Ruta de navegación

La **ruta de navegación** define el camino que un vehículo recorre con el objetivo de llegar a su destino. Es representada por un subconjunto de segmentos de carretera adyacentes entre sí, los cuales forman un camino continuo. El primer elemento se trataría del inicio de la ruta y el último, el destino.

Este objeto se define como una colección ordenada de objetos, denominados **Fragmentos de ruta** *(RouteFragments)*. Éstos representan fragmentos de la ruta que transcurren en un segmento de carretera determinado. A su vez, estos están compuestos por un par de objetos que representan las posiciones inicio y fin de estos fragmentos. Éstos se han denominado **Puntos de ruta**. (RoadPoint)

El navegador del vehículo es el responsable de generar la estructura de la ruta y de gestionar el avance del vehículo pos sus posiciones mediante los atributos “posición” y “puntero de la ruta” del navegador.

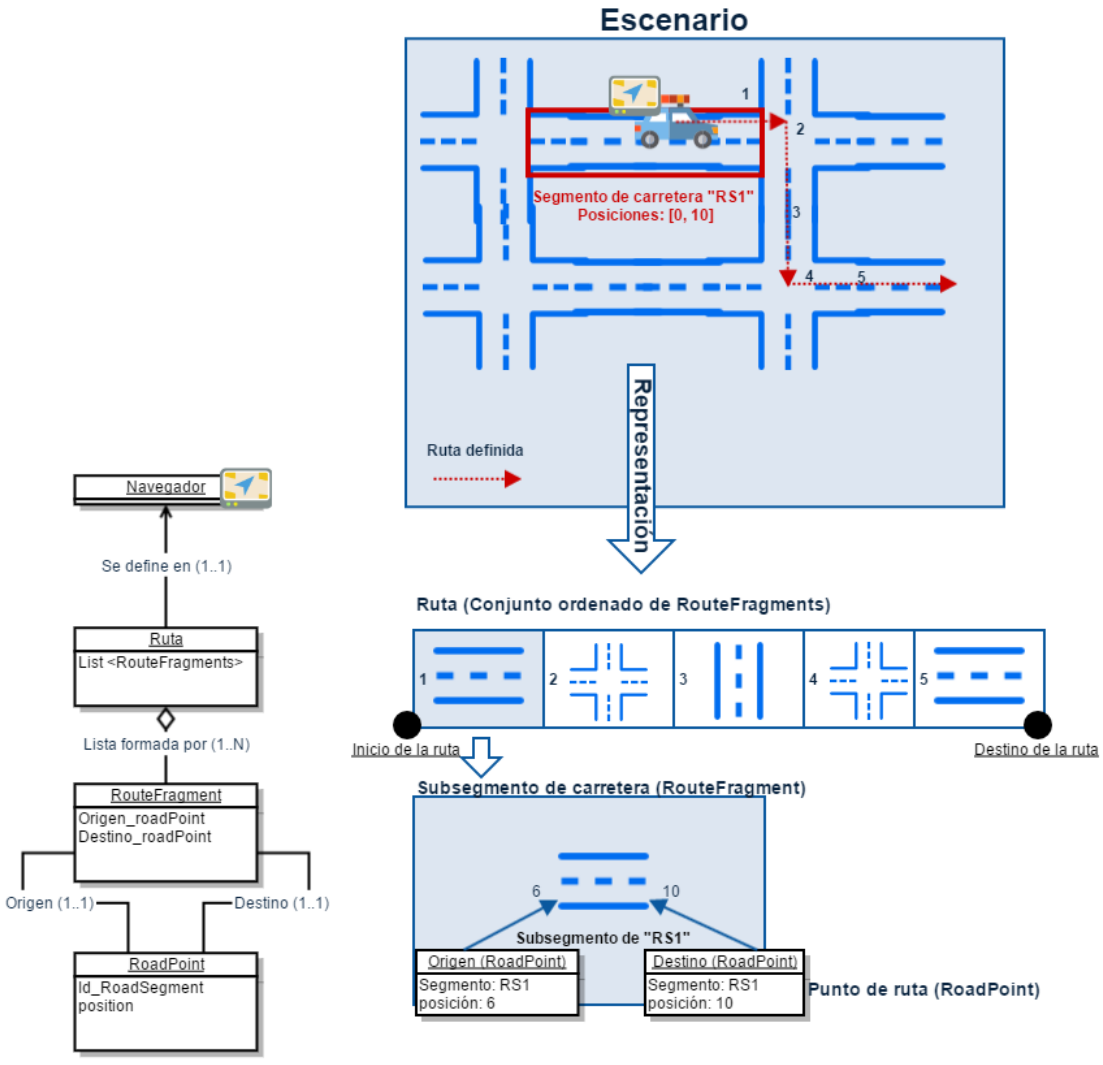

Figura 6: Diagrama de clases de "Ruta" y escenificación gráfica

4.6. Señalizaciones de tráfico

Con los objetos de las carreteras y los vehículos con navegación ya dispondríamos de componentes suficientes para simular un entorno de tráfico, pero sin ningún tipo de regulación ni contratiempo, simplemente los vehículos podrían ir del inicio de su ruta a su fin sin ningún control ni obstáculo. Para ello se diseñan los componentes de señalización, semáforos, señales e incidentes. Con estos componentes se consigue una aproximación más real a un entorno urbano que queremos conseguir.

El modelado de estos elementos pretende ser una correspondencia con sus homólogos en el mundo real y afectar al tráfico de vehículos en diferentes posiciones de las carreteras. Aunque el objetivo sea el mismo para todos ellos, su configuración es distinta y representan eventos bien diferenciados, por lo que se describen a continuación de manera independiente.

Todos tienen la característica que tienen que comunicar su estado a la carretera asociada para que esta pueda establecer su estado y el vehículo al consultarlo pueda determinar su siguiente posición en el siguiente paso. A continuación, se describe el comportamiento general de cada uno.

4.6.1. Señales de tráfico

Representa al conjunto de las señales de tráfico existentes en el mundo real, estas pueden afectar al tráfico de diferentes maneras, por ejemplo, pueden regular la velocidad máxima de un tramo de carretera o impedir el paso de cierto tipo de vehículos. Están compuestas por su **condición** y por estar ligadas a una **carretera** y a una **posición inicial y final** que representan su posición y el conjunto de posiciones en las que hace efecto su condición.

4.6.2. Semáforos

Su funcionamiento es idéntico al de su homólogo en el mundo real. Mediante **3 estados** distintos (Representadas por las luces *verde, ámbar y roja*) controlan el flujo de vehículos que atraviesan su **posición**, permitiendo o denegando el paso por ella. Para el escenario solo se han tomado en cuenta los estados verde y rojo que representan el permiso y la denegación de circulación. Estos estados se deben ir turnando cada cierto tiempo a lo largo de la simulación para que los vehículos afectados por el estado rojo no estén detenidos eternamente frente al semáforo. Por lo tanto, éstos tienen que ser gestionados en cada paso de simulación para que se vayan alternando sus estados. Este comportamiento dinámico los diferencia de las señales y los incidentes.

Los semáforos tienen la peculiaridad de tener que comunicar a la carretera a la que pertenecen su estado cada vez que actualicen para que ésta pueda proporcionarlo a los vehículos y éstos actúen en consecuencia.

4.6.3. Incidentes de tráfico

Son eventos inesperados que cortan el flujo en un conjunto de posiciones de una carretera. Representan incidentes de diferentes categorías con el efecto común de impedir el paso. Por ejemplo, podría ser un choque entre vehículos, un árbol caído o cualquier otro evento de que pudiera ocurrir en un escenario real. Tienen como atributos las **posiciones** que están afectadas, el **tipo de incidente**, que representa el evento al que representa y su **estado**, que significa si está activo o no (llega un momento en el tiempo en que el incidente es subsanado).

Figura 7: Escenificación de Señales, semáforos e incidentes

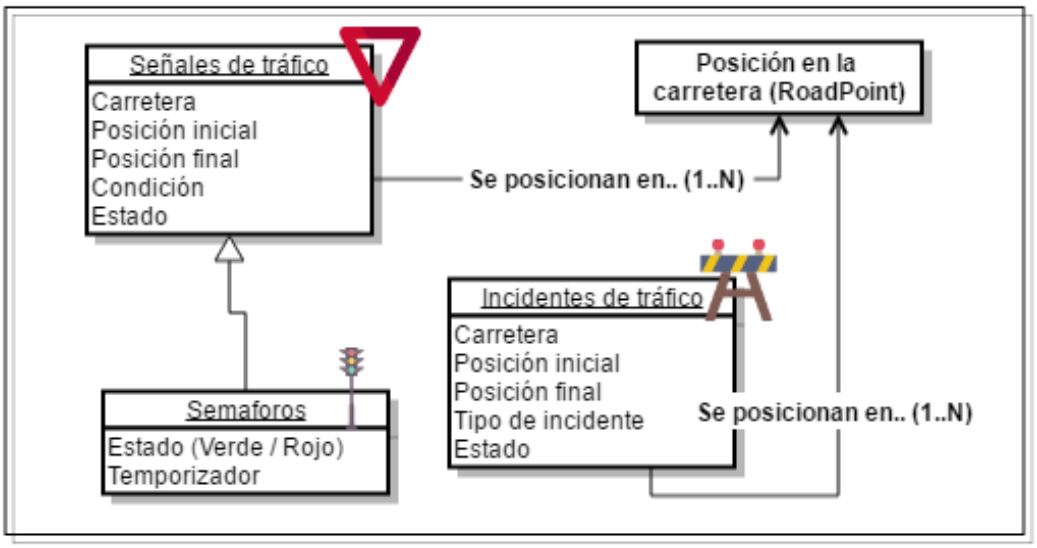

Figura 8: Diagrama de clases "Señalización de tráfico"

5. Diseño

5.1. Introducción

En este capítulo se muestran las decisiones de diseño que se llevan a cabo para crear la infraestructura del sistema y la integración de los componentes dentro de esta. Además, se recalcan los patrones de diseño más relevantes y se da una visión más expandida de la relación entre las clases de los componentes.

5.2. Arquitectura del sistema

A partir de los componentes analizado y las clases extraídas se pasa a diseñar la infraestructura general del sistema y la posición de cada componente en este. Dando especial relevancia a las comunicaciones entre módulos internos y con los usuarios.

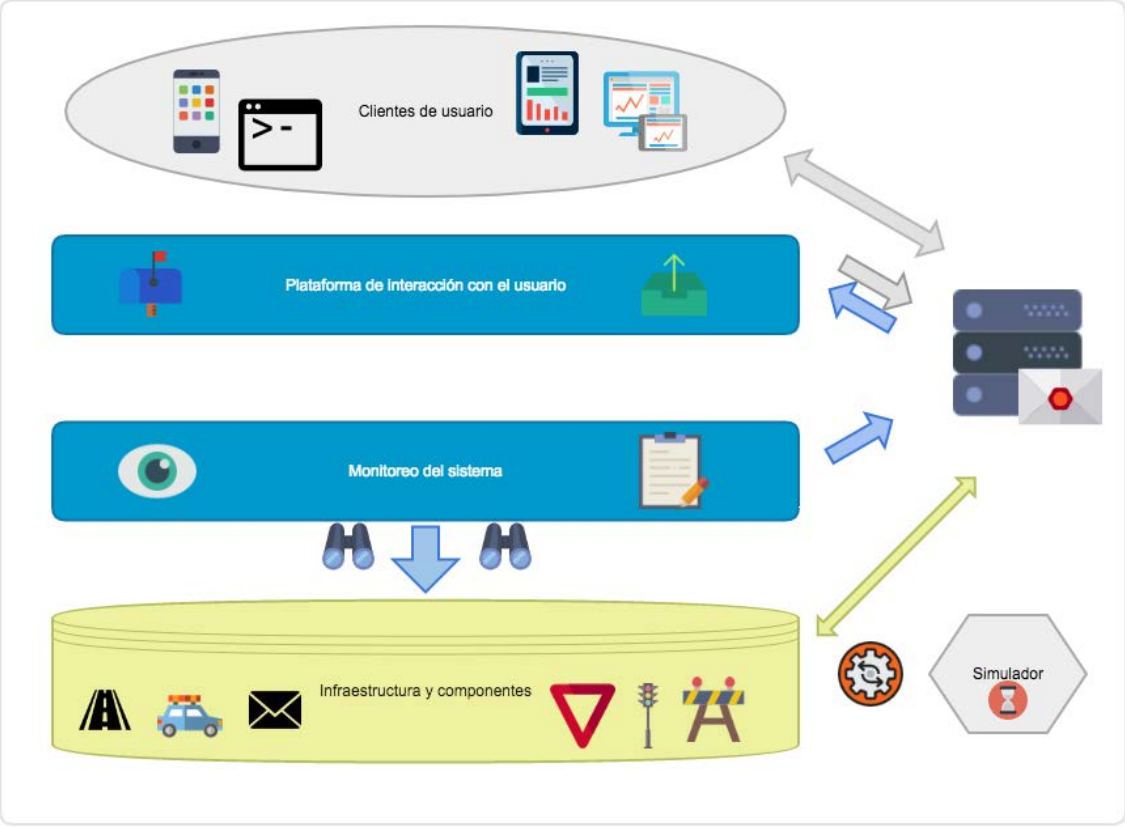

Figura 9: Arquitectura del sistema

Los componentes de la anterior figura se pasan a comentar en los siguientes apartados del presente capítulo.

5.2.1. Componentes transversales del sistema

Los presentes componentes tienen la característica común de que proporcionan su funcionalidad al conjunto del sistema. Se trata, del simulador y del servicio de comunicaciones.

- **El Simulador**, representado en la figura como el hexágono gris, es el encargado de la gestión temporal del escenario y de emular el control de los elementos del sistema en cada paso de emulación. Dentro de sus funcionalidades destacan la inicialización del escenario en el sistema, establecer un orden de ejecución de cada elemento del escenario y el paso de control en cada turno del control para que cada elemento realice las comunicaciones necesarias con lo demás para actualizar su propio estado y notificar del mismo.

- **El servicio de comunicaciones**, representado en la figura como el rack de servidores, es un servicio que da soporte a las comunicaciones de todos los componentes del sistema, incluyendo a los usuarios. Su funcionalidad se basa en la gestión de manera eficiente de todos los canales de comunicación necesarios para que los componentes inteligentes del sistema puedan comunicarse entre ellos. Éstas comunicaciones están representadas en la figura por las flechas bidireccionales de cada color. Está compuesto por un servidor accesible por todo el sistema donde los componentes pueden subscribirse y publicar mensajes a diferentes temáticas establecidas.

5.2.2. Infraestructura de los componentes de tráfico

A continuación, se presentan los componentes del sistema de tráfico como resultado del análisis del escenario propuesto. Está representado en la figura como el cilindro amarillo y contiene todos los elementos que representan el sistema de gestión autónoma de tráfico.

- **El tramo de carretera** (*RoadSegment)*, incluye toda la lógica necesaria para representar una versión simplificada del estado de su homologo real (capacidad máxima, número de vehículos, velocidad máxima, longitud, etc..) además de poder intercomunicarse con los otros componentes para compartir dicha información.

- **El vehículo** (*vehicle)*, incluye información de estado y de su navegación como su velocidad, ruta y posición. El objetivo del vehículo es seguir la ruta definida reaccionando con los eventos del contexto y llegar a su punto de destino. Para ello debe poder comunicarse con otros componentes para avisar de su movimiento y poder adaptarse a los eventos que vayan ocurriendo que pudieran afectarle (reducir velocidad, pararse ante un semáforo, detenerse ante un accidente, etc.).

- **El semáforo** (*TrafficLight)*, define un comportamiento independiente que simula el funcionamiento real de un semáforo. Tiene que ser capaz de comunicar rápidamente su estado al contexto ya que este afectará a los vehículos que quieran circular por dicha posición. Impidiendo el paso si el semáforo está en rojo y permitiéndolo en cualquier otro estado. Comportamiento similar a un semáforo real.

- **Las señales de tráfico** (*TrafficSignals)*, se encargarán de la regulación del tráfico al igual que los semáforos, por lo que su funcionamiento es similar pero su estado es más estático. Por ejemplo, podría existir una señal en una carretera que impidiera el paso o regulara el paso de vehículos.

- **Incidentes de tráfico** (*TrafficIncidents)*, simulan el evento de un accidente de tráfico genérico y tienen como finalidad interrumpir la circulación del tráfico.

5.2.3. Infraestructura “Human in the Loop”

Uno de los objetivos importantes del sistema es proporcionar los mecanismos necesarios para que el usuario participe de manera activa a la hora de la toma de decisiones del sistema autónomo. Para ello se ha diseñado una arquitectura independiente del sistema de gestión de tráfico que sea el encargado de la gestión de los eventos que son comunicados al conjunto de los usuarios.

Por ejemplo, es el encargado de que avise a un conductor de que una de las carreteras de su ruta está congestionada. Para ello es necesario el diseño y desarrollo de una capa encargada de la monitorización del sistema que recoja los eventos definidos, otra capa para la gestión de la interacción con el usuario que retransmita el evento monitorizado a los usuarios finales y una aplicación independiente que permita la interacción del usuario con el sistema.

- **Monitorización del sistema,** representado en la figura por el rectángulo azul inferior con el icono del ojo, es el encargado de, mediante el despliegue de una serie de sondas de monitorización, monitorice los valores de los atributos de los *Smart Objects* aprovechando que éstos tienen la capacidad de compartir dicha información con el sistema. Para ello se necesita la definición genérica de las sondas de monitorización y su aplicación en el desarrollo de una sonda por evento monitorizado. Para el escenario se definen dos tipos de monitorización, para la notificación de alta densidad de carreteras y para notificar incidentes de tráfico.

- **Plataforma de interacción con el usuario,** representado en la figura por el rectángulo azul superior con el icono del *mailbox*, es el encargado de recoger los eventos notificados por la monitorización y establecer mediante el usuario o rol asociado a la configuración de la sonda, quienes van a ser el usuario o grupo de usuarios que van a recibir el mensaje. Además, es el encargado de formatear el mensaje que se reproducirá en los clientes de usuario y de realizar dicho envío.

- **Cliente de usuario,** representado en la figura por el conjunto superior de dispositivos digitales**,** es la interfaz de usuario encargada de mostrar las notificaciones del sistema ofreciendo también la capacidad de poder enviar respuesta de vuelta al sistema para su procesamiento. Debido a la variedad de roles de usuario el cliente debe de ser capaz de parametrizarse dependiendo del usuario y su rol, para así ser útil a todos los usuarios del sistema.

5.3. Patrones de diseño

La finalidad de esta sección es la de explicar las decisiones de diseño realizadas y defender los motivos principales tenidos en cuenta. Para el diseño del sistema se toman en consideración las características principales de autonomía del sistema y de la independencia entre los componentes. Esto implica que sean independientes entre ellos y tengan las mínimas dependencias cruzadas. Además, es necesario disponer de una API de comunicación entre componentes para su interacción.

Con esta independencia y autonomía se pretende además incluir transparencia en las implementaciones internas de los módulos para que su modificación no tenga repercusión en el sistema. Esta característica es fundamental para su futura implementación en entornos reales, que es también otro objetivo del proyecto.

Por último, también se expone el diseño de la infraestructura dedicada a la participación del usuario en el sistema, teniendo éste un papel relevante.

A continuación, se exponen los diseños de los patrones más relevantes empleados y la infraestructura de los módulos que componen el sistema.

5.3.1. Componentes transversales del sistema

Durante el proceso de análisis del proyecto se ha podido observar que la mayoría de componentes presentan estructuras similares que se podían modelar con patrones comunes, por lo que se opta por diseñar y estandarizar dichas estructuras mediante patrones homogéneos para su uso en cada componente. La decisión se basa en que el sistema está diseñado para ser fácilmente extendido y en un futuro añadir nuevos módulos y funcionalidades, es por ello que es fundamental la estandarización en el diseño.

Los patrones más relevantes empleados transversalmente en el proyecto se exponen en los siguientes apartados.

5.3.1.1. Configurador de un objeto

Este patrón se caracteriza por la inclusión a un objeto de una clase asociada a la que se ha denominado **configurador** que tiene como objetivo el ser una clase auxiliar donde se almacenan y consultan aquellos atributos propios del objeto asociado que no se espera que cambien de valor durante la ejecución, es decir, que sean de valor **estático**. Manteniendo los de carácter dinámico dentro del objeto principal.

Dentro del configurador de una carretera se podrían incluir atributos como la longitud o la capacidad máxima de la misma, valores que una vez establecidos en el inicio de ejecución, no varían en ejecución. Este patrón ha sido utilizado dentro de los objetos de la infraestructura de los elementos simulados (vehículos, carreteras, señalización...)

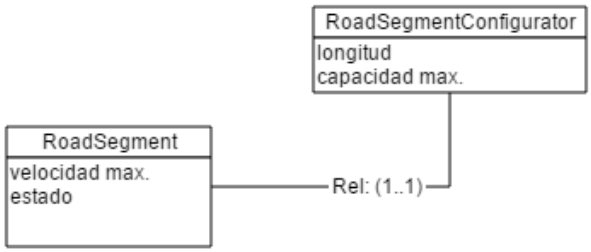

Figura 10: Ejemplo del configurador del RoadSegment (carretera)

5.3.1.2. Comunicador (Publicador y Subscriptor)

Las comunicaciones entre los servicios tienen un papel fundamental y deben de realizarse de manera desacoplada e independiente a la implementación interna del objeto. Para ello se propone el desarrollo de un patrón de diseño que permita acceder a los métodos de la API de comunicaciones del objeto de manera transparente a la implementación de los métodos y al modo de comunicación, para que en un futuro si se cambia de modelo de mensajería, solo hubiera que cambiar la implementación de esta API.

Esto se logra mediante un patrón de clases denominado **Comunicador** que hace transparente el acceso a dos subclases que son las que implementan los métodos de la API de publicación y de subscripción. Este patrón se repite en todos los componentes que para su funcionamiento deben de comunicarse con otros.

Por ejemplo, en el caso de los vehículos, la API estaría formada por una clase Comunicador que abstraería los métodos de las clases publicadora y subscriptora del vehículo. En la publicadora estarían los métodos para comunicarse el vehículo con el medio, como por ejemplo notificar que entra o sale de una carretera, y en la subscriptora se encontrarían los métodos para subscribirse y anular subscripción a los eventos de una carretera específica.

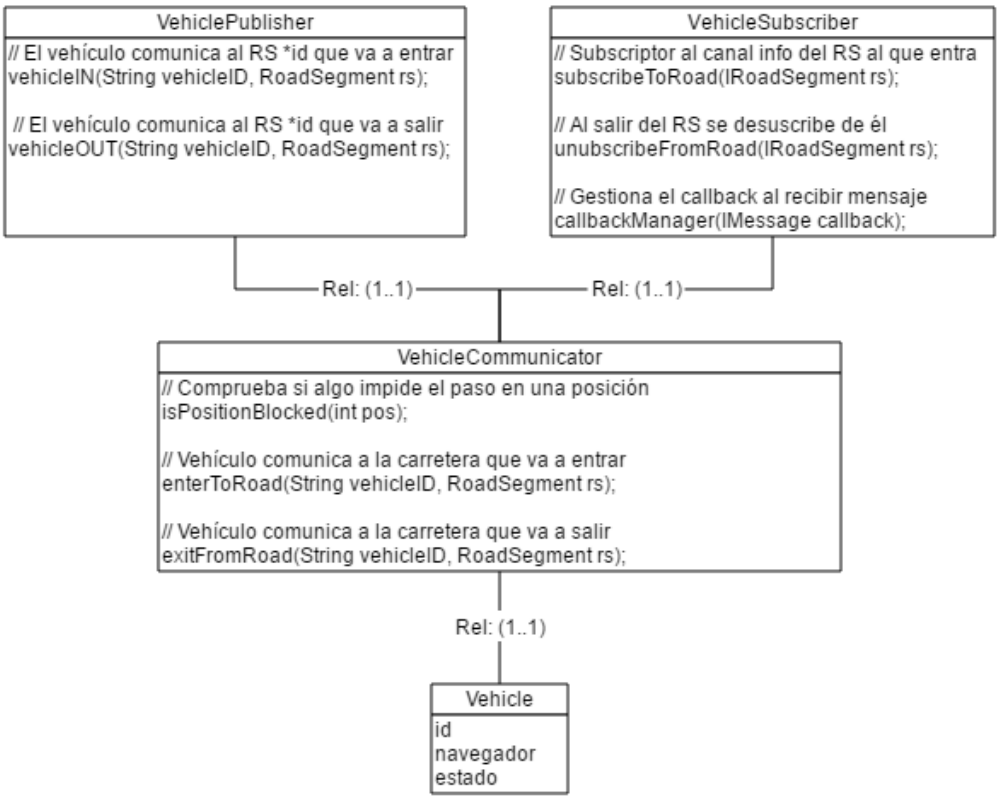

Figura 11: :El Comunicador del vehículo

Como puede verse en la figura, la clase comunicador sería el API del vehículo que proporciona una serie de métodos que a su vez llaman a las implementaciones en las clases de publicación y subscripción Así por ejemplo, si un vehículo quiere avisar de su salida de una carretera usaría su método "*exitFromRoad*" que a su vez ejecutaría el método del publicador "*vechicleOUT*" y se suscribiría de la nueva carretera, cancelando la anterior, con los métodos de la clase subscriptora "*subscribeToRoad*" y "*unsubscribeFromRoad*" . Los detalles de las API de los componentes se explican con detalle en el capítulo de implementación.

5.3.2. Diseño del simulador

Para poder implementar los escenarios virtuales donde ejecutar y probar el sistema propuesto, es necesario el diseño de un componente que se encargue de la gestión de todos los demás y de alguna manera llevara el control de éstos. Este componente se ha denominado el **simulador**.

El simulador es el elemento central del sistema ya que es el encargado de instanciar todos los elementos y de ir gestionando sus pasos de ejecución a través del tiempo. Se trata del único componente del sistema que no ha sido totalmente diseñado para el presente proyecto, si no que ha sido importado de otros proyectos anteriores donde fue también necesario el uso de un simulador de un entorno compartido formado por diversos componentes independientes. El utilizar un simulador importado ha influido en el diseño de las comunicaciones de los componentes y en la elección de las tecnologías utilizadas en la implementación.

El diseño del simulador debe cumplir las siguientes funcionalidades para poder ofrecer un entorno de simulación funcional a sus componentes:

- Definición de los parámetros de simulación
- Instanciación del simulador
- Gestión del ciclo de vida del simulador
- Registro de los componentes de simulación
- Instanciación de las estancias de los componentes
- Gestión del ciclo de vida de los componentes
- Funcionalidad de cada componente al realizar la actualización
- Actualización de estado periódica del sistema

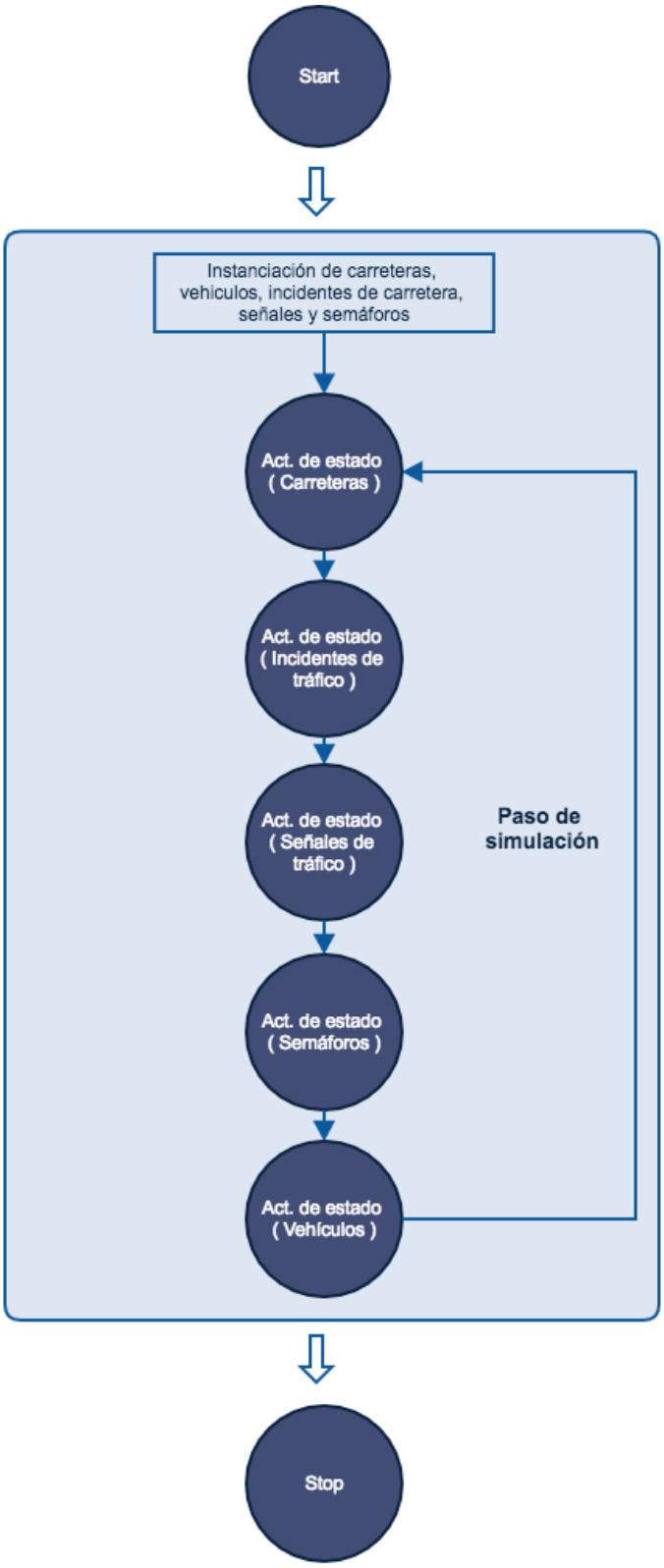

Figura 12: Ciclo de vida del simulador

5.3.3. Diseño de las API de comunicaciones en el dominio de tráfico

Uno de los objetivos principales es el de proporcionar un sistema de regulación de tráfico totalmente autónomo e inteligente, donde los componentes interactúen entre si y tomen decisiones de manera propia a medida que avanza el tiempo. En este capítulo se pasa a describir el diseño de las estructuras de comunicaciones de los componentes previamente analizados. Siempre con la visión de la compatibilidad futura en un escenario real. Es por ello que muchas decisiones de diseño han sido tomadas para que el cambio entre los entornos tenga el mínimo coste.

Ya que un entorno de tráfico real es altamente dinámico y caótico, ha sido necesario que nuestro sistema no fuera centralizado y fuera altamente escalable, por lo que tener un servidor central con toda la información del contexto no era una opción a tomar en cuenta. Por lo tanto, se ha optado por que cada componente tenga una visión parcial del sistema y que tengan que comunicarse entre ellos para obtener la información puntual relevante. Motivo por lo que se hacía necesario una API de comunicaciones que les permitieran compartir información y gestionar las peticiones y respuestas.

A continuación, se exponen los detalles de diseño de los modelos de comunicación y las funcionalidades que aportan al sistema.

5.3.3.1. Segmento de Carretera (RoadSegment)

Las carreteras son responsables de comunicar al entorno de su estado de congestión de tráfico y para gestionarlo deben de tener un control de los vehículos que las están recorriendo en un momento dado.

Para ello se ha diseñado la estructura "**Comunicador**", descrita en la sección 5.3.1.2. Mediante esta estructura, las carreteras se pueden subscribir y publicar asíncronamente los siguientes datos:

- Encargado de publicar:
    - Su estado en todas las posiciones de la carretera (por ejemplo, carretera cerrada)
    - Su estado en una posición determinada de la carretera (por ejemplo, accidente en la posición 3)
- Se subscribe a los siguientes eventos:
    - Vehículos que entran en la carretera
    - Vehículos que abandonan la carretera

5.3.3.2. Navegador

Las comunicaciones del vehículo se integran en su navegador asociado y permiten obtener en tiempo real toda la información necesaria para que este pueda decidir la siguiente posición del vehículo, además de informar de ciertos eventos a los demás elementos de simulación. El patrón de comunicación es de tipo publicador y subscriptor y su objetivo es obtener y enviar los siguientes datos:

- Encargado de publicar:
    - La entrada a una carretera
    - La salida de una carretera
- Se subscribe a los siguientes eventos:
    - Si puede avanzar a la siguiente posición de su ruta. En caso de no poder tendría que pararse en la posición anterior.

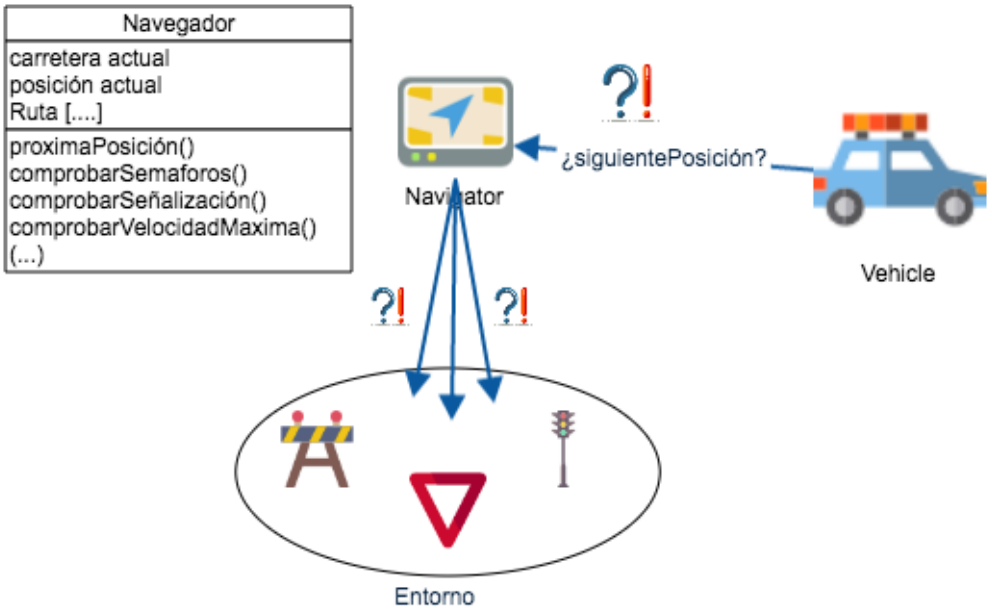

Figura 13: Escenificación del navegador

5.3.3.3. Gestor de tráfico

Una carretera tiene que ser capaz de llevar un control de los vehículos que la cruzan, para ello es necesario de alguna estructura de datos que de manera dinámica se encargue de almacenar dicha información y proporcione una API de gestión para simplificar su uso.

En el presente módulo se almacena y gestiona dicha información recibida por el componente de comunicaciones. En el caso de la carretera, en éste se guarda el contador o listado de los vehículos que están recorriéndola en un momento dado.

Esta información es almacenada en una estructura de datos gestionada por una clase gestora asociada a cada carretera. La información que guarda es de carácter volátil ya que cada vez que un vehículo entra y sale de la carretera, ésta tiene que actualizar el contador de su capacidad.

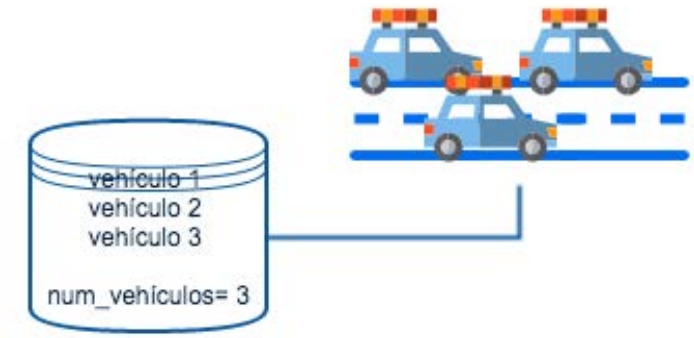

Figura 14: Escenificación del gestor de tráfico

5.3.3.4. Gestor de incidencias

De manera similar a la carretera con el gestor de tráfico, el vehículo necesita almacenar cierta información del entorno para poder formarse de una imagen del estado de la carretera que está circulando para saber si puede avanzar o no. Esta información necesita de alguna estructura de datos no persistente, ya que solo es útil durante un corto periodo de tiempo, pudiendo ser consultada rápidamente, ya que su información es crucial para el movimiento del vehículo. Dicha estructura se ha diseñado como una clase auxiliar dependiente del objeto navegador a la que se ha denominado **Gestor de incidencias**.

En el presente módulo se almacena y gestiona la información recibida por el componente de comunicaciones. En el caso del vehículo, en éste se guarda el estado de las incidencias de la carretera en que está circulando en cada momento. Estas incidencias incluyen los estados de los semáforos y cualquier otro evento que obstaculice la circulación por una posición de la carretera.

Esta información es almacenada en una estructura de datos gestionada por una clase gestora asociada a cada navegador. La información que guarda es de carácter volátil ya que, al cambiar de carretera, el navegador limpia la estructura, ya que la información ya no le es relevante.

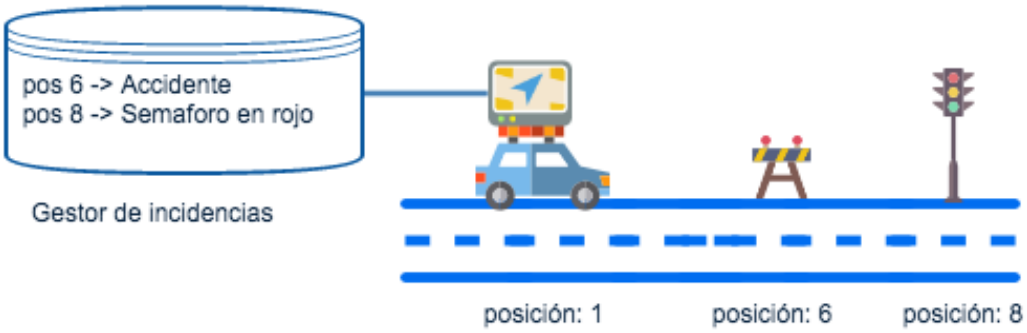

Figura 15: Escenificación del gestor de incidencias

5.3.4. Diseño de los componentes “Human in the Loop”

En esta sección se trata el diseño de todos los componentes que forman parte de la capa que integra a los diferentes usuarios como un componente más del sistema. Estos componentes se han diseñado de manera paralela al sistema gestor de tráfico compartiendo sistema de comunicaciones con él y con los usuarios.

5.3.4.1. Monitorización

Uno de los objetivos principales del proyecto es el de proporcionar información del sistema al usuario. Para la obtención de dicha información se ha diseñado una capa independiente y externa al funcionamiento del simulador y de sus componentes que se encargue de comprobar y transmitir la información de interés de los elementos fuente. Esta nueva capa se ha denominado la capa de **monitorización,** ya que como indica su nombre es la encargada de monitorizar a los componentes sin que estos sean conscientes y transmitir la información al usuario.

El diseño de esta capa se ha realizado mediante la creación de una clase abstracta denominada **monitor** de la que extienden todos los **monitores especializados** que se necesiten implementar. Cada monitor especializado tratará un objeto distinto y filtrará por diferentes atributos, éstos se verán más en detalle en el capítulo de implementación, dejando en el aparado actual de diseño la explicación de su clase abstracta.

El funcionamiento base de la clase monitor es:

- **Subscripción**: El monitor se subscribe a cierto evento informativo de cualquier elemento que proporcione la información.
- **Procesamiento del evento**: Al recibir el evento este es procesado por el monitor. Este procesamiento tiene que ser implementado en cada monitor y dependerá del objetivo del mismo.
- **Publicación**: Si la información del evento es relevante se pasa a la publicación de la misma en forma de aviso o petición de feedback a la capa de interacción de usuario para que informe o solicite información.

Además, está compuesto de diferentes atributos que indica a quienes tiene que redirigir dicha información la capa de interacción con el usuario. Esta capa se describe en la siguiente sección.

- **Rol**: Representa el tipo de usuario al que está dirigido el evento. Para el escenario propuesto se han diseñado 2 tipos, el **conductor del vehículo** y el **agente de tráfico**. Cada rol estará interesado en diferentes tipos de eventos.
- **Usuario**: Representa a un usuario individual. Puede ser, por ejemplo, el conductor de un vehículo determinado o un agente de tráfico de la ciudad.

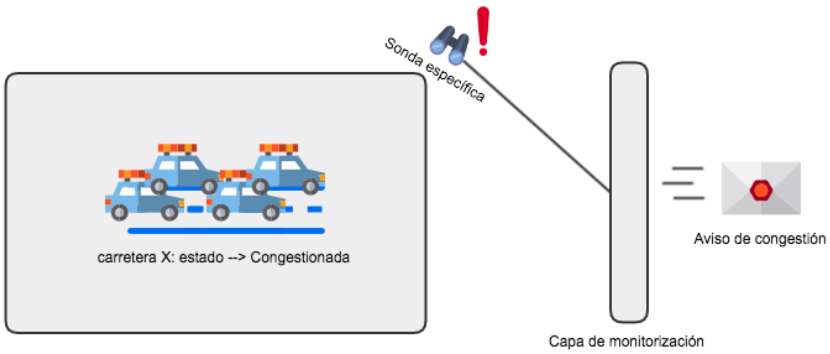

Figura 16: Escenificación de la monitorización

5.3.4.2. Capa de interacción con el usuario

Uno de los objetivos principales es que el usuario pudiera formar parte del sistema, recibiendo información relevante para él, o proporcionando información que pudiera ser relevante al sistema para realizar informes o gestionarse internamente. Para ello se ha creado la capa de monitorización que está formada por diferentes monitores que reaccionaban a partir de un evento determinado y a su vez lo retransmitían con un rol y usuario objetivo a una capa superior que es la encargada de decidir a qué usuarios finales enviar los mensajes.

Esta nueva capa, intermedia al sistema de monitorización y al usuario final, se le ha denominado **Interacción con el usuario** o "*User Integration*" (UI). Esta capa es la encargada de recibir los eventos de los monitores y junto a la meta información de los roles y usuarios adscritos, formar un mensaje orientado a la presentación en un cliente de usuario. La finalidad de esta capa es por tanto doble, formatear los mensajes para la comunicación a humano, ya que en las demás capas del sistema son comunicaciones M2M (Machine to Machine) y enviar a los usuarios finales dicho mensaje.

Las comunicaciones, como en las demás capas, son de tipo publicador y subscriptor y éstas se caracterizan por la recepción de los eventos monitorizados y el envío a usuarios y a roles finales de mensajes ya formateados para su presentación.

- Encargado de publicar:
    - Eventos monitorizados por todos los monitores inicializados en el sistema.
- Se subscribe a los siguientes eventos:
    - Mensajes orientados al usuario de tipo notificación y solicitud de *feedback* a un usuario.
    - Mensajes orientados al usuario de tipo notificación y solicitud de *feedback* a un rol (conjunto de usuarios).

5.3.4.3. Cliente de usuario

Por último, para que los usuarios puedan interactuar fácilmente con el sistema, ha sido necesario el diseño de una interfaz donde poder recibir la información de los eventos del mismo y poder comunicarse con él. Esta interfaz debe de ser independiente totalmente del sistema y poder ser configurable para poder servir a diferentes tipos de usuarios sin necesidad de fragmentar funcionalidades en diferentes clientes.

Para el diseño del cliente se han tenido en cuenta las peculiaridades que tiene dentro del sistema. Se trata de un elemento no necesario para el funcionamiento del sistema del cual se puede conectar y desconectar cuando quiera. Además, su interacción con el sistema no debe de ser bloqueante, el usuario puede perfectamente omitir la información proporcionada y no responder, aunque se le solicite. De este comportamiento se han extraído las siguientes funcionalidades que debe cumplir:

- Escucha de un canal definido por el rol y el usuario donde se subscribe al instanciarse y donde recibe todos los mensajes dirigidos a dicho rol y usuario.
- Al recibir un mensaje muestra su contenido por la interfaz y si es un mensaje de tipo *feedback* proporciona al usuario una serie de opciones definidas dentro del mensaje para que pueda responder si lo desea.
- En caso de feedback, al seleccionar una opción el usuario, el cliente envía la respuesta de vuelta a la capa de interacción con el usuario del sistema para que la tenga en cuenta.

Este cliente puede implementarse de muchas maneras; aplicación independiente, por consola de comandos, etc. Para el escenario propuesto se ha optado por una aplicación nativa para dispositivos Android debido a su popularidad y facilidad de uso entre otros motivos.

6. Implementación

6.1. Introducción

En el presente capítulo se describen los detalles de la fase de implementación de todos los módulos del sistema y de sus compontes. Se explica, tanto el proceso de desarrollo del sistema como las decisiones tomadas para la transformación de los diseños expuestos a las estructuras de código final del sistema. El presente capítulo se estructura de manera análoga a los paquetes de código desarrollados, por lo que los nombres de los componentes y los diagramas de clases son los mismos que los presentes en la implementación del sistema.

6.2. Estructura del proyecto (módulos desarrollados)

La estructura de la implementación realizada corresponde con las diferentes capas de las que se compone el sistema comentadas previamente. Cada una es desarrollada de manera independiente en paquetes de código distinto y con las mínimas dependencias cruzadas. Como se ha comentado previamente, este era uno de los objetivos del proyecto. Por tanto, se le ha dado mayor visibilidad a las diferentes API de comunicaciones de cada componente.

Los módulos (o capas) en cuestión son:

- **Simulation**: Incluye todas las clases e interfaces de definen un simulador genérico y a sus herramientas. Es utilizado como base para la creación del simulador dentro del módulo S4CSmartTraffic. Por lo que se trata de una dependencia a dicho módulo.

- **S4CMessaging**: Incluye todas las clases e interfaces de la infraestructura de mensajería que se ha empleados a lo largo de todo el proyecto. Incluye las jerarquías de los diferentes tipos de mensajes empleados y su formato. Es un módulo trasversal a los módulos que necesitan comunicarse con otros, por lo que se trata de una dependencia a ellos, estos son los cuatro siguientes módulos.

- **S4CMonitoring**: Incluye todas las clases e interfaces que definen e implementan la capa de monitorización al gestor de tráfico. Incluye la implementación de un monitor abstracto y todos los monitores específicos implementados.

- **S4CSmartTraffic**: Incluye todas las clases e interfaces de los componentes y de la implementación del escenario planteado. Este módulo formaría el conjunto de la infraestructura de simulación en su totalidad.

- **S4CUserIntegration**: Incluye todas las clases e interfaces necesarias para la implementación de la capa externa del sistema que conecta al cliente de usuario con el sistema. Es el encargado del envío de las notificaciones y de la gestión del feedback del usuario.

- **Cliente de usuario**: Incluye todas las clases, interfaces y librerías necesarias para la implementación de una interfaz de usuario con dos funcionalidades principales, notificar al usuario destinatario de los mensajes del sistema y proveer la funcionalidad de poder responderlos. Se trata de una implementación Android nativa, por lo que incluye todos los paquetes de clase propias de una aplicación Android, además de las clases propias del desarrollo de las funcionalidades mencionadas. En este documento nos centraremos únicamente en las clases propias a las funcionalidades implementadas.

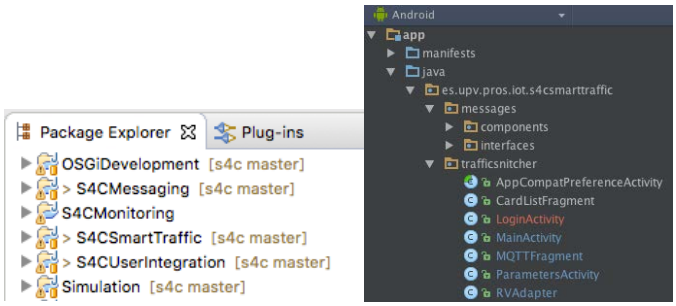

Figura 17: Módulos dentro del workspace del sistema y cliente de usuario

6.3. Comunicación contextual

Debido a que los componentes de simulación del tráfico deben de estar totalmente desacoplados para simular un comportamiento real, estos tienen que incluir un mecanismo para ser instanciados independientemente y poder comunicarse y compartir un contexto común. Para el escenario simulado, ésto se consigue utilizando el framework OSGi, que nos proporciona las herramientas necesarias para desplegar las instancias de dichos componentes como servicios independientes simulando un contexto común para todas ellas y que estas se pudieran comunicar entre sí.

Las comunicaciones entre los diferentes componentes de tráfico se realizan mediante el uso de un contexto común que nos ofrece el *framework* OSGi. En este contexto, que todos los servicios (*bundles* en OSGi) comparten, se pueden guardar, consultar y modificar diversos objetos. Esto nos permite compartir información entre los servicios y tener una visión global del sistema, manteniendo todas las partes independientes entre sí sin saber la existencia unos de otros.

Este tipo de comunicación compartida entre *bundles* es especialmente importante en el gestor de tráfico ya que, para simular un entorno de tráfico real, los componentes se tenían que comportan de manera individual sin conciencia de los demás, y para comunicarse debían de utilizar algún tipo de API para comunicarse entre ellos. El *BundleContext* cumple con dicha función en el presente proyecto como se verá en las siguientes secciones del presente capítulo.

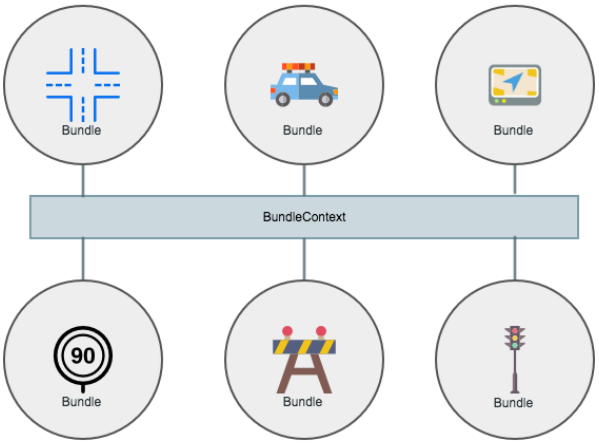

Figura 18: OSGi bundles y BundleContext

6.4. Comunicación entre módulos (APIs de interoperabilidad)

Siguiendo con el enfoque del sistema distribuido, es necesario el desarrollo de una infraestructura de comunicaciones para el intercambio de información entre los diferentes módulos del sistema, además de para comunicarse con el usuario. Para ello se emplea la tecnología MQTT de IBM que nos proporciona las herramientas necesarias para comunicar, de manera asíncrona, los diferentes módulos del sistema utilizando un patrón de mensajería publicador / subscriptor.

Esta tecnología se ha utilizado en las implementaciones de las API de comunicaciones de los componentes del sistema. Para el desarrollo del proyecto y el escenario se ha utilizado MQTT ya que está orientada a proyectos IoT y nos permite el paso de mensajes entre módulos de manera sencilla solo necesitando un servidor (*broker*) que gestione el paso de mensajes. Queda resaltar que la arquitectura del proyecto permitiría hacer cambio de tecnología teniendo solo que modificar el código pertinente dentro de las API de comunicaciones, siendo transparente al resto del sistema. Tecnologías como REST [18] o zeroMQ [19] serían otros posibles candidatos.

Para la implementación realizada con MQTT se necesita el despliegue de un servidor MQTT que es el encargado de realizar todas las peticiones de subscripciones y publicaciones mediante el uso de diferentes *topics* o canales definidos mediante una cadena de texto, como por ejemplo "/s4cui/user/{id-rol}/{id-user}/", siendo los dos primeros atributos fijos y los otros dos, patrones dinámicos dependientes de los valores de rol y usuario en tiempo de ejecución.

El uso de esta tecnología se utiliza principalmente en las API de comunicaciones de los componentes de los módulos dentro de los patrones implementados de tipo *Comunicador* descrito en el capítulo de diseño. Los detalles de cada API se describen en la sección de cada componente en el presente capítulo de implementación.

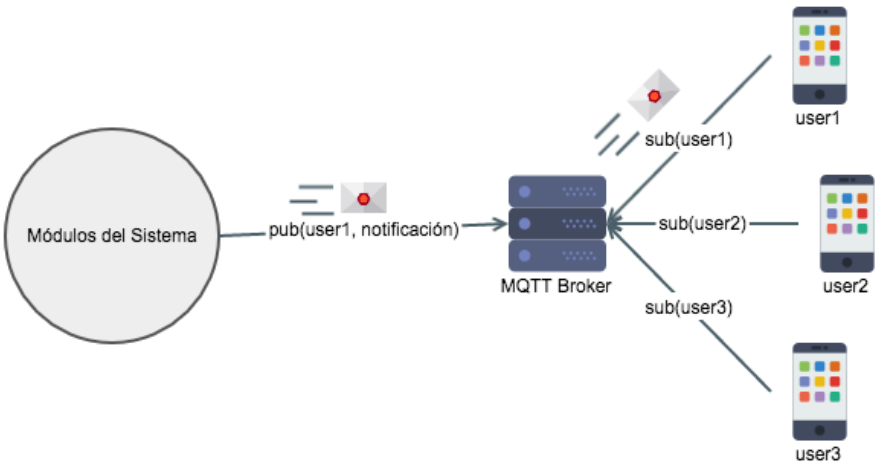

Figura 19: Ejemplo de comunicaciones pub/sub con MQTT

6.5. Infraestructura de mensajería (S4CMessaging)

Para la mensajería del proyecto se utilizan 2 tipos de formatos de mensajes diferenciados para las comunicaciones internas y externas del sistema. Para comunicaciones M2M, entre los módulos del sistema, se ha denominado ***"Message"*** y entre sistema y usuario, ***"UserMessage"***. Todas las clases de mensajería internamente son representadas en formato JSON por su fácil implementación en proyectos Java. Se implementan en estas clases, constructores Java a partir de un objeto JSON y conversores del objeto Java a JSON para la gestión de estos objetos para la creación y manipulación de los mensajes.

A continuación, se describen los detalles de implementación de las clases de mensajería empleadas.

6.5.1. Message

Clase abstracta de la que extienden todos los mensajes internos al sistema. Incluye todos los atributos y métodos básicos de los mensajes.

- Identificador
- Timestamp
- Origen (Source)
- Tipo (Clase del mensaje)

Tipos de mensajes internos:

- **TrafficMessage**: Formato utilizado para los mensajes de tipo tráfico. El emisor es una instancia de un vehículo y el emisor es una instancia de la clase carretera. La finalidad es informar el vehículo a la carretera de diferentes eventos definidos por "*ETrafficMessageType*". Para el escenario propuesto implementan los casos "VEHICLE_IN" y "VEHICLE_OUT" que indican la entrada y salida de un vehículo en la carretera.
- **InfoMessage**: Formato utilizado para los mensajes de tipo información. El emisor es una instancia de una carretera y el emisor, una instancia de un vehículo. La finalidad es la de informar de los eventos de tráfico que pueden influir en el estado de circulación del vehículo, tales como cambio de estado de un semáforo o un incidente de tráfico.


```
{"infoType":"TRAFFICLIGHT_CHANGE","pos_fin":30,"roadID":"S2","id":"S2-1473431515680",
"source":"S2","type":"INFO","pos_ini":30,"timestamp":"2016-09-09
16:31:55.68","status":"1"}
```


Figura 20: Ejemplo de "Message", (cambio de estado de un semáforo)

Está compuesto por los siguientes atributos:

- RoadID: Identificador de la carretera.
- Sition_ini: Posición inicial del evento.
- Position_fin: Posición final del evento.
- InfoType: Tipo de mensaje de información. Representado por EInfoMessageType.
  - TRAFFICLIGHT_CHANGE
  - TRAFFICACCIDENT
  - TRAFFICSIGNAL_CHANGE
  - INCIDENT_TYPE
  - TRAFFIC_CONGESTION
  - ROAD_RESET
  - OTHER
- Status: Representa si impide o no la circulación por las posiciones afectadas de la carretera.


```
{"options":"{\"0\":\"Aceptar\",\"1\":\"Cancelar\",\"2\":\"Otros\"}","id":"m22016-09-0
9 16:29:07.668-1473431405208","type":"USERFEEDBACK","message":"Carretera [S2] estado:
Saturated","user":"c1","rol":"VEHICLE_DRIVER","timestamp":"2016-09-09 16:30:05.208"}
```


Figura 21: Ejemplo de "UserMessage", (Petición de feedback)

6.5.2. UserMessage

Clase abstracta de la que extienden todos los mensajes orientados al usuario. Incluye todos los atributos y métodos comunes a todos los mensajes extendidos.

- Identificador del mensaje.
- Timestamp: Indica el momento exacto de la emisión del mensaje.
- Rol: Indica al rol del grupo de usuarios al que está destinado el mensaje.
- Usuario: Indica el identificador único de un usuario.
- Message: Texto en claro que contiene la información del usuario

Tipos de mensaje de usuario:

- **UserNotificationMessage**: Formato utilizado para mensajes de usuario de tipo notificación. Estos mensajes cuentan con los mismos atributos que la clase abstracta *UserMessage* y son utilizados por el sistema para, únicamente, informar al usuario de eventos a los que esté subscritos.

- **UserFeedbackMessage**: Formato utilizado para mensajes de usuario de tipo feedback. Estos mensajes extienden el formato de mensajes de usuario añadiendo la funcionalidad de poder responder a dichas notificaciones. Incluyen los siguientes atributos:
    - User: Usuario destinatario de la petición de feedback
    - Options: Objeto JSON con las opciones y sus valores internos (var, value) entre las que el usuario puede reaccionar ante la notificación.

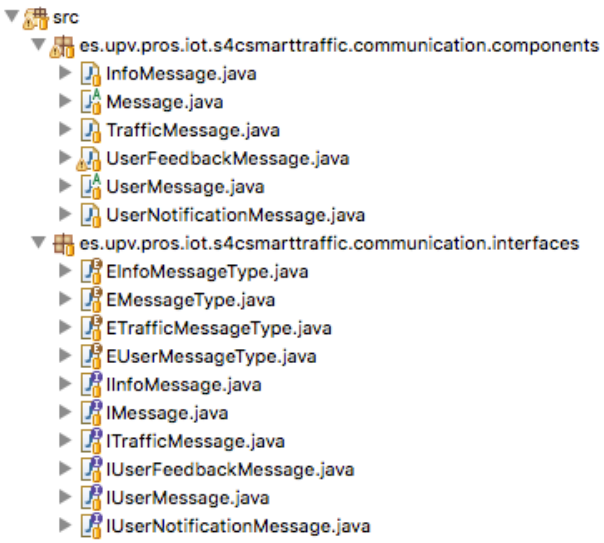

Figura 22: Arquitectura del módulo de mensajería

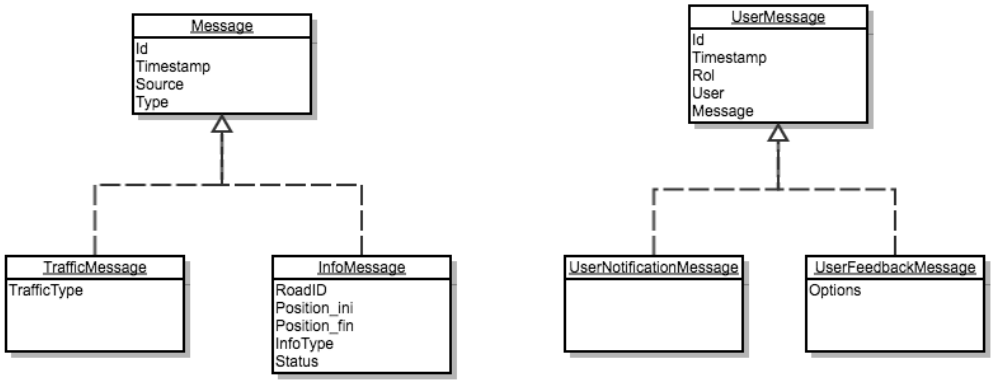

Figura 23: Jerarquía de los tipos de mensajes

6.6. Sistema para la gestión de tráfico (S4CSmartTraffic)

En este capítulo se describe la implementación de todos los componentes del módulo de gestión del tráfico, además de la implementación del escenario simulado propuesto que se describirá en el capítulo del prototipo. El objetivo es el de tener un sistema autónomo de gestión de tráfico simulado funcional. Para ello, el módulo se ha implementado en 5 distintos paquetes principales descritos a continuación.

- **Componentes de simulación**: Se definen las clases de los componentes que forman parte de la simulación. Dichas clases están formadas por los atributos propios de los componentes y la API de métodos que se ofrece para su gestión. Además, incluye los métodos para instanciarlo como una *bean* en el contexto OSGi y asociarles la API de comunicaciones (patrón comunicador) para las comunicaciones por mensajería, MQTT.

Los principales componentes son:

- **Vehículo** (*Vehicle*): Representación de un vehículo real con los siguientes atributos que permiten su configuración y que se gestionan como propiedades de su OSGi *bean*. Estos son los siguientes:
  - Navegador: Objeto *Navigation* ligado al vehículo en la que se delegan todas las funcionalidades del vehículo ligadas con la navegación por las carreteras.
  - Velocidad: Indica la velocidad del vehículo en un momento determinado.
  - Estado: Indica el estado del vehículo, puede ser "En circulación o *Moving*", "Parado o *Stopped*" o "Estacionado o *Parked*".

- **Navegador** (*Navigator*): Objeto asociado a un vehículo en la que se delega toda la funcionalidad para la gestión de rutas y movimiento del mismo durante la simulación. Su funcionalidad es la de ofrecer al vehículo una API de métodos que le permita la gestión de su movimiento de manera trasparente a la implementación de la ruta y a la de su gestión interna. Dicha API está compuesta por numerosos métodos, ya que el movimiento del vehículo es complejo y debe comprobar numerosas variables internas y del entorno para decidir la siguiente posición en la ruta. Los más relevantes son:

  - void nextRouteStep(): Establece la siguiente posición del vehículo basándose en la posición y la velocidad del vehículo. Además, gestiona la entrada y salida entre varios *RoadSegments*.

- RouteFragment getRouteStep(): Devuelve el *RouteFragment* actual de la ruta.

- int getNextPosition(): Devuelve la siguiente posición del vehículo basándose en velocidad y posición del vehículo y comprobando que no haya obstáculos en las posiciones intermedias. En caso de haberlas devuelve la posición inmediatamente anterior al obstáculo.

- boolean finalPointerPosition(): Confirma si el vehículo se encuentra en el último tramo de su ruta.

- void nextRoutePointer(): Avanza el puntero de la ruta hacia el siguiente tramo.

- boolean checkObstaclesAtPosition(RoadSegment rs, int pos): Comprueba si existe algún obstáculo en el RoadSegment y posición indicados.

- void enterToNextRoadSegment(): Gestiona la entrada del vehículo al siguiente RoadSegment.

- void exitFromCurrentRoadSegment(): Gestiona la salida del vehículo del RoadSegment actual.

- boolean hasArrived(): Confirma si el vehículo se encuentra en destino.

- int checkObstacles(int pos, int nextPos): Comprueba si hay obstáculos entre un tramo de posiciones concretos.

- boolean isRouteStart(): Comprueba si el vehículo se encuentra en el inicio de la ruta.

- void startRoute(): Realiza la instanciación del vehículo en su ruta, extrapolándolo a un escenario real representa la entrada del vehículo en su ruta.

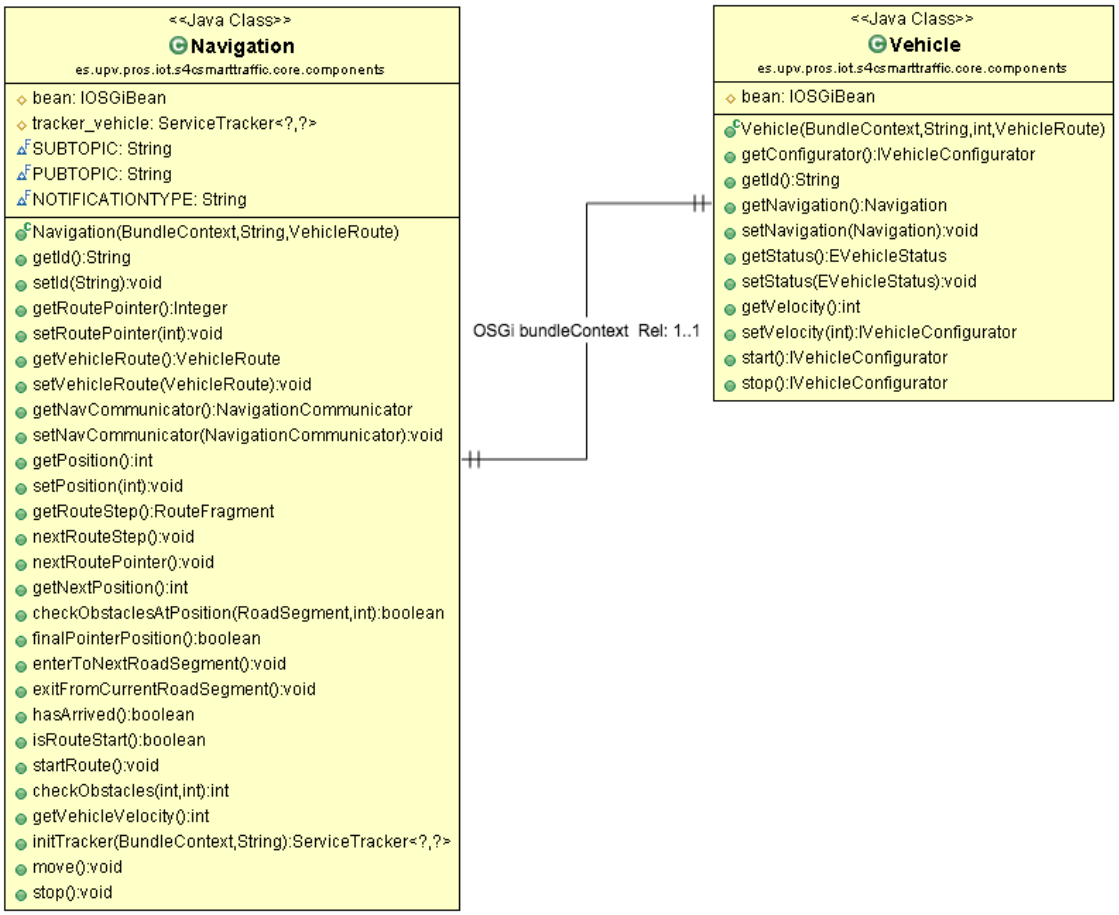

Figura 24: Diagrama de la relación Vehículo-Navegador

- **Ruta** (*VehicleRoute*): Representa una ruta de tramos de carreteras que se asocia al navegador de un vehículo. La implementación se ha realizado mediante un *ArrayList* de componentes *RouteFragments*. Dichos fragmentos representan los tramos de carreteras de la ruta, representados a su vez como una tupla de *RoadPoints*, que es a su vez una representación puntual en el espacio de una carretera, por ejemplo (RS1, 5) representaría un *RoadPoint* de la carretera "RS1" y su posición 5.

- **Tramo de carretera** (*RoadSegment*): Representa a una carretera real. Su configuración está formada de identificador, longitud y capacidad y cuenta como atributos dinámicos, su velocidad máxima y su estado (Fluido, Denso, Saturado). Además de los métodos en su API para consultar dichas propiedades, incluye la funcionalidad de informar a la capa de monitorización de los incidentes de tráfico. Este método sería el enlace de comunicación entre el simulador y el monitor de incidentes de tráfico.

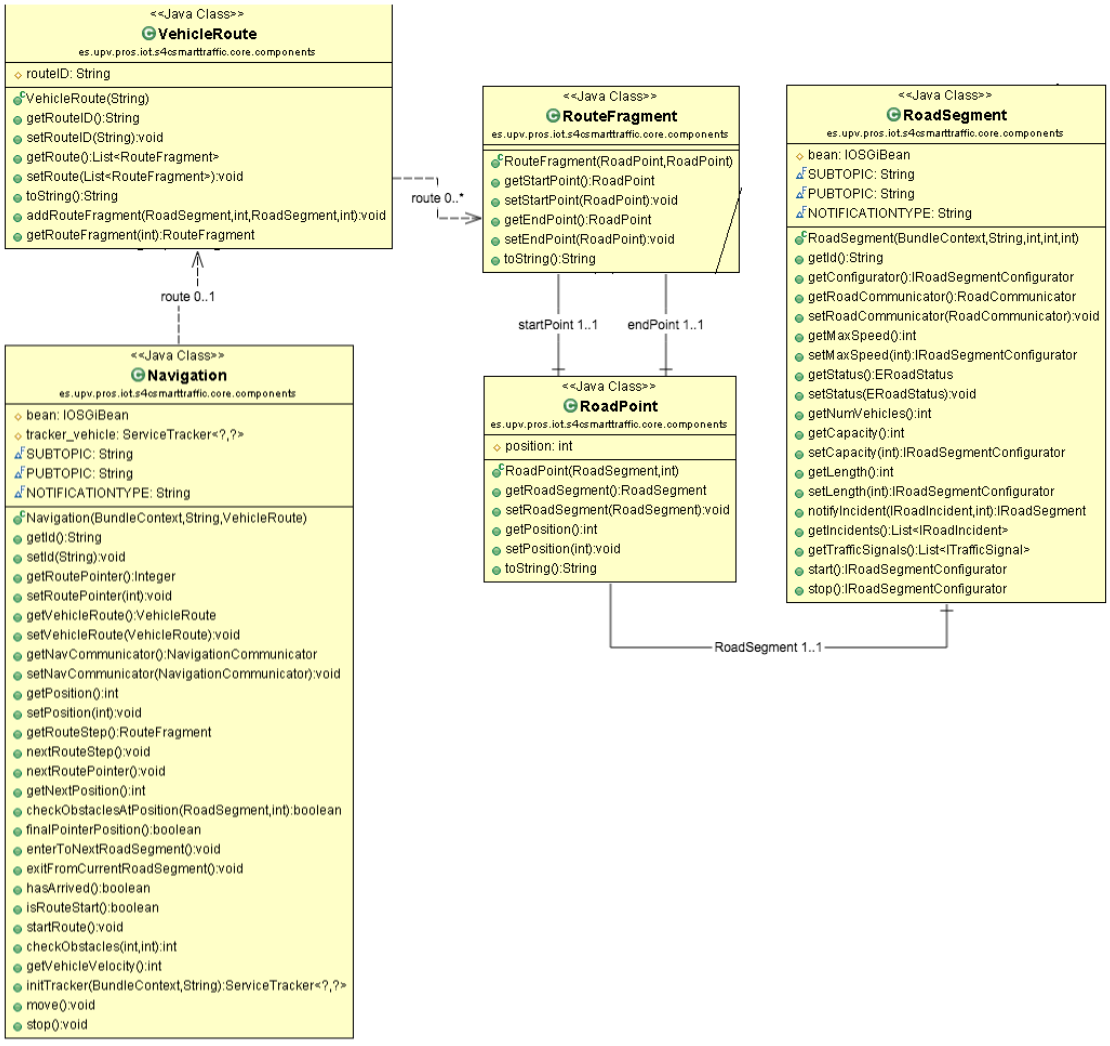

Figura 25: Diagrama de clases de VehicleRoute y RoadSegment

- **Señal de tráfico** (*TrafficSignal*): Representación de una señal de tráfico y compuesta básicamente de un identificador, el tipo de señal de tráfico, el área de efecto mediante una posición inicial y final, el *RoadSegment* donde se encuentra y el estado que afectan a los vehículos en el área de efecto.

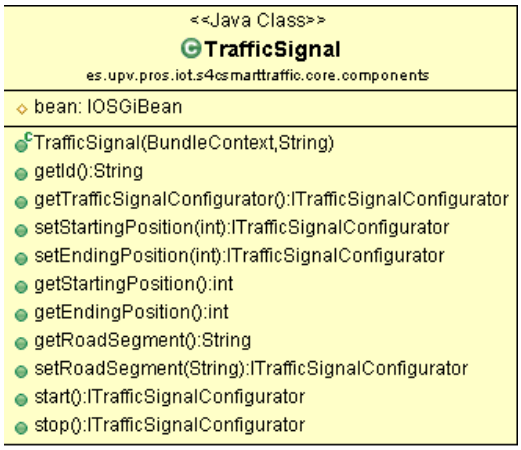

Figura 26: Diagrama de clases TrafficSignal

- **Incidente de tráfico** (*RoadIncident*): Representación de un incidente temporal en un tramo de posiciones de una carretera. Su implementación es muy similar a la de las *TrafficSignals* con la peculiaridad que estas se pueden activar y desactivar representando que dichos incidentes ocurren y son solucionados en un periodo determinado de tiempo.

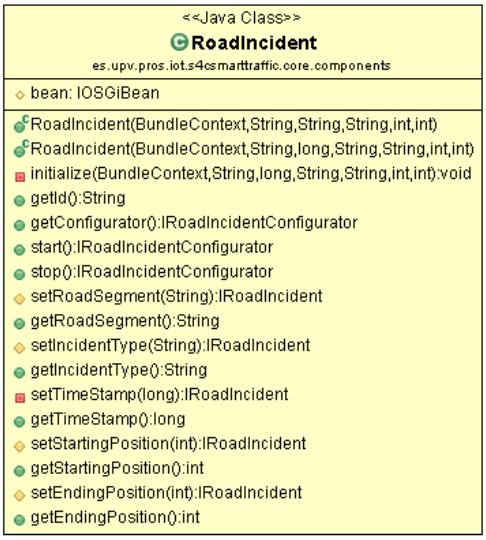

Figura 27: Diagrama de clase RoadIncident

- **Semáforo** (*TrafficLights*): Representación de un semáforo. Está compuesta por varias clases, la clase principal que incluye los atributos y la API, un controlador donde se configuran todos los parámetros de funcionamiento y el *worker* que gestiona, con la información del *controller*, la temporización y el cambio de estado automático de las luces del semáforo. La implementación realizada agrupa varios de ellos en dos grupos distintos y así poder gestionarlos de forma agrupada pudiendo tener valores distintos en el mismo paso de simulación en los distintos grupos. La API de la clase tiene los métodos necesarios para obtener el estado actual del semáforo y obtener su configuración.

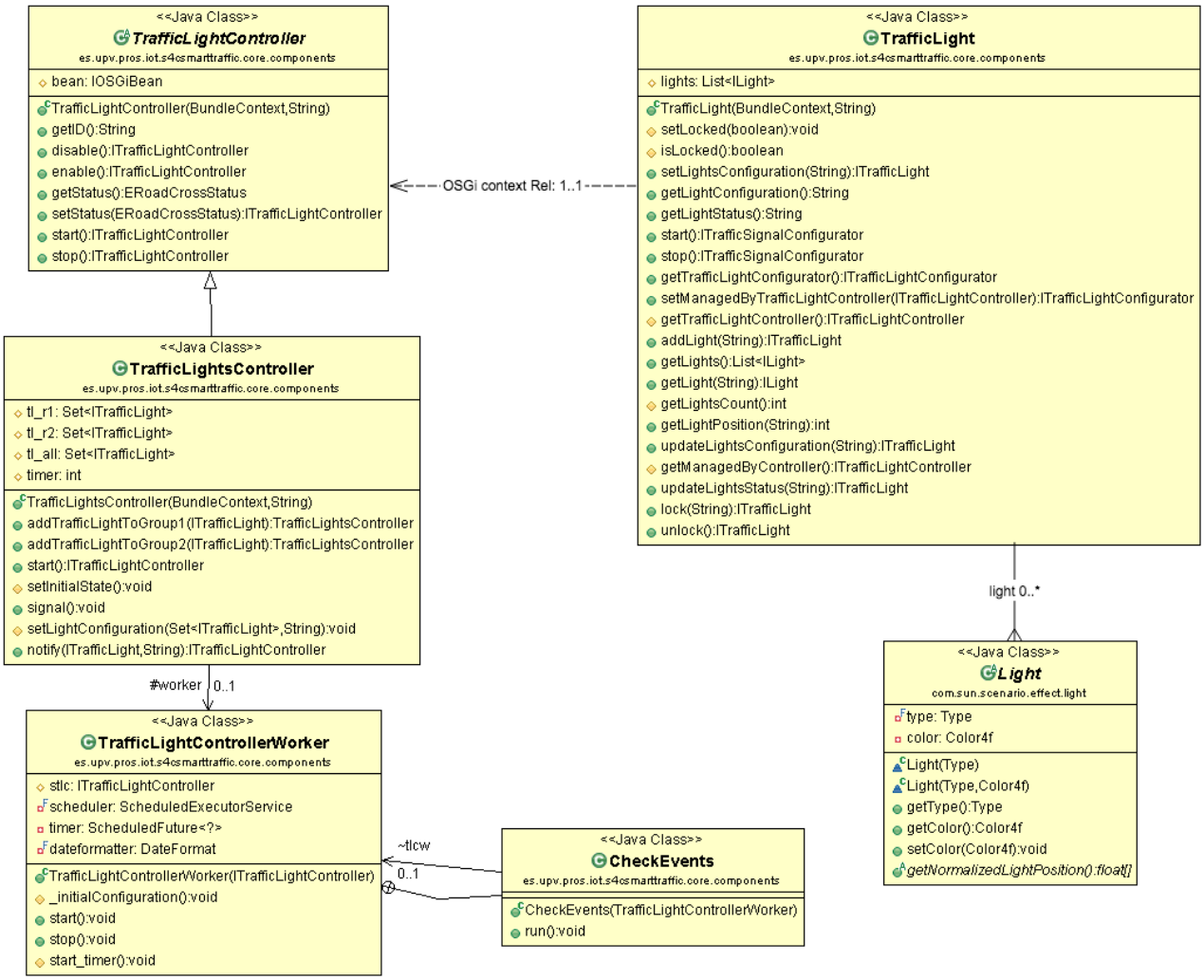

Figura 28: Diagrama de clases de TrafficLights con sus componentes y controlador

- **Componentes de comunicación**: Estos componentes son las clases auxiliares que implementan la API de comunicaciones de los componentes siguiendo el patrón **Comunicador** con la finalidad de proporcionar una serie de métodos para comunicarse con los demás. Los principales componentes que utilizan este patrón son los *RoadSegments* y los *Navigation*, esto es, las carreteras y los navegadores de los vehículos.

    - **NavigationCommunicator**
      Comunicador asociado al navegador de un vehículo que ofrece diferentes métodos para interactuar el vehículo con su entorno, especialmente con las carreteras.

        - isPositionBlocked(final int pos): El vehículo solicita información sobre el estado de una posición para saber si puede atravesarla.

        - enterToRoad(final String vehicleID, final RoadSegment rs): Realiza la subscripción al siguiente RoadSegment de la ruta y le informa de que va a entrar. (Publicando un mensaje de tipo vehicleIN)

        - exitFromRoad(final String vehicleID, final RoadSegment rs): Anula la subscripción al canal del actual RoadSegment y le informa de que va a salir. (Publicando un mensaje de tipo vehicleOUT)

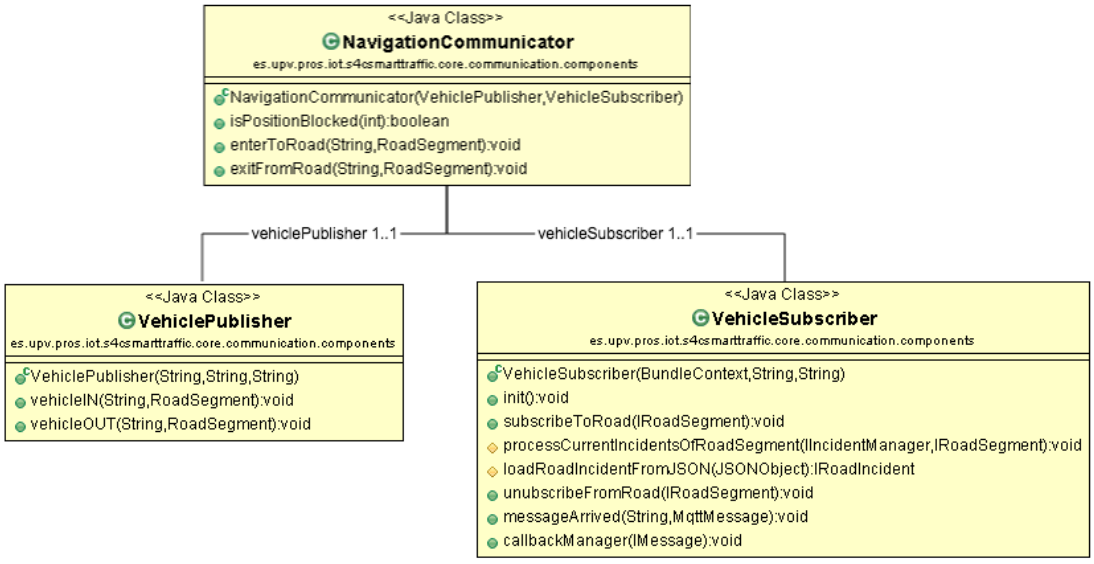

Figura 29: : Diagrama de clase NavigationCommunicator

- **RoadCommunicator**
  Comunicador asociado a un objeto RoadSegment que le proporciona diferentes métodos para interactuar con los vehículos.

  - updatePositionStatus(final String roadID, final int pos_ini,final int pos_fin, final EInfoMessageType type, final String status): Actualiza, en el *dataManager* de un *RoadSegment*, el estado asociado a un intervalo de posiciones.

  - int getNumVehiclesInRoad(): Obtiene del *trafficManager* el número de vehículos que circulan en un momento dado por el *RoadSegment*.

  - sendRoadStatus(final String roadID, final String status): Publica la actualización del estado del tráfico de un *RoadSegment* al canal del módulo de monitorización.

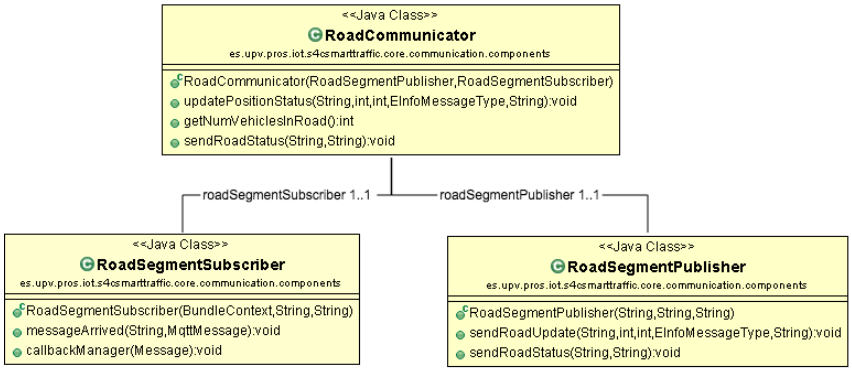

Figura 30: Diagrama de clases RoadCommunicator

- **Data Managers de los componentes**: Estas clases son el resultado de la necesidad de gestión de información dinámica proveniente del entorno por parte de los componentes. Tanto los vehículos como las carreteras necesitan de un componente que gestione la información que se comparten entre sí. Este componente es el denominado **dataManager** y cumple con la funcionalidad de proveer una API para la gestión de la información de contexto almacenada en cada instancia de ambos objetos.

  - **TrafficManager**:

    Gestiona el contador del número de vehículos en un *RoadSegment* para obtener la densidad de tráfico a tiempo real. Está implementado como un *array* de **vehicleID** y su longitud es el contador.

  - **IncidentManager**:

    Gestiona el listado de posiciones de un *RoadSegment* que presentan un obstáculo. Esta estructura es la que los vehículos consultan para saber si en una posición determinada existe algún obstáculo que le impide avanzar. Esta está implementada como un *HashTable* donde la *key* seria la **posición** y el *value* el **estado** asociado a dicha posición. Con esta estructura se pretende crear un método ligero y rápido para la consulta de los estados, ya que es una información crucial para el correcto funcionamiento de los vehículos y necesita ser escalable al número de vehículos.

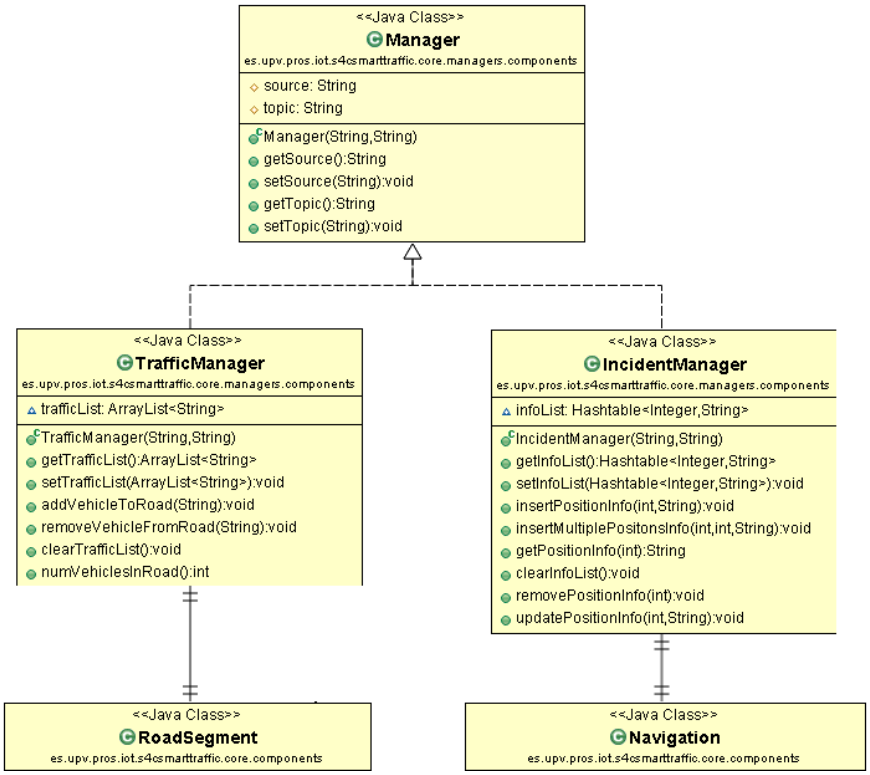

Figura 31: Diagrama de clases de los dataManagers

- **Simulation Element Managers** (SEM): Estas clases implementan los métodos que ejecutan el comportamiento de los componentes en cada paso de simulación. Estos métodos son ejecutados en cada paso de simulación dentro de la clase SEM, **SmartTrafficSimulator**, en él se definen de todos los pasos que tienen que realizar cada instancia de cada componente del sistema. Cada módulo se va actualizando siguiendo el orden establecido hasta que se detiene el sistema o se paran sus componentes.

  - **VehiclesSEM**

  Se encarga en grandes rasgos del comportamiento del movimiento del vehículo. Siguen los siguientes pasos:

  - Si es el primer ciclo, se coloca al vehículo en la ruta
  - Si no:
    - Se comprueba si el vehículo ha llegado a destino, en caso afirmativo se cambia su estado a estacionado (Stopped)
    - Se mueve el vehículo a la siguiente posición de su ruta "vehicle.getNavigation().move()". Internamente se realizan las comprobaciones del entorno para determinar la siguiente posición. Para el SEM el funcionamiento interno del método "move()" es totalmente transparente, se delega todo al navegador.

  Además, para dotar al sistema de simulación mayor autonomía se ha implementado un módulo de creación de instancias de vehículos cada cierto tiempo con configuraciones de navegación definidas. Con eso conseguimos que no comiencen todos los vehículos en el momento cero de la simulación y se pueden definir diferentes rutas de navegación para que no sigan todos los vehículos las mismas rutas. Este módulo es el denominado *VehiclesRandomGenerator* y se encarga tanto de la generación aleatoria de vehículos como de incidencias, como se indica en el apartado de *RoadIncidentSEM*.

- **RoadSegmentsSEM**:

  En la actualización de las carreteras se realizan las siguientes tareas:

  - se mira el número de vehículos en cada instante y se va actualizando el contador de tráfico y su estado siguiendo el siguiente algoritmo:
    - Tráfico → (numVehicles * 100) / CapacidadMáxima; Estado → (Tráfico < 50 = Fluido; > 50 & < 75 = Denso; > 75 & < 99 = Saturado; 100 = Cerrado)

  - En el caso de que cambie el estado a estar saturado o deje de estarlo, se manda una notificación, vía la API de comunicación, al sistema de monitorización para que este alerte a los usuarios afectados. Corresponde con el funcionamiento del *trafficMonitor* ya mencionado.

  - Se actualizan los estados de los elementos de tráfico (updateTrafficSignals, updateTrafficLights(...))

- **SmartTrafficSimulator**

Integra el funcionamiento del método *callback* ejecutado en cada paso de simulación. En su método *takeSimulationStep* se buscan todos los componentes registrados en el contexto OSGi de cada tipo y se ejecuta el método *takeSimulationStep* de su SEM.

- **TrafficLightsSEM**

Realiza la actualización de los valores de las luces de los semáforos. Es el encargado, tomando como analogía un escenario real, de cambiar las luces del propio semáforo. Para conseguir un funcionamiento más realista de los semáforos, estos se han dividido en dos grupos con configuraciones distintas, esto quiere decir que existen dos funcionamientos distintos para los semáforos, pudiendo estar en estados distintos y teniendo una temporización entre cambios de estado diferentes.

- **TraficSignalsSEM**

Realiza las actualizaciones de valores de las *TrafficSignals* de manera similar a los vehículos y las carreteras. Debido a que solo se han implementado *TrafficSignals* de tipo estáticas, no se realiza ninguna acción en el *takeSimulationStep* pero en caso de añadir señales dinámicas, la arquitectura ya está desarrollada.

- **RoadIncidentsSEM**

Realiza las actualizaciones de los valores de los incidentes de tráfico, que simbolizarían la aparición y la solución de los mismos. Para dotarle de un funcionamiento realista se ha implementado conjuntamente de un generador y cerrador de incidentes de tráfico aleatorio. Mediante una probabilidad definida, cada paso de simulación se evalúa y en caso de acierto se genera y registra una incidencia en una serie de posiciones contiguas pseudo-aleatorias de una carretera. Teniendo el mismo funcionamiento su cierre, en caso de acierto se cierra el incidente más antiguo. Esta funcionalidad se encuentra en el módulo VehiclesRandomGenerator que también se encargaba de la generación automática de vehículos.

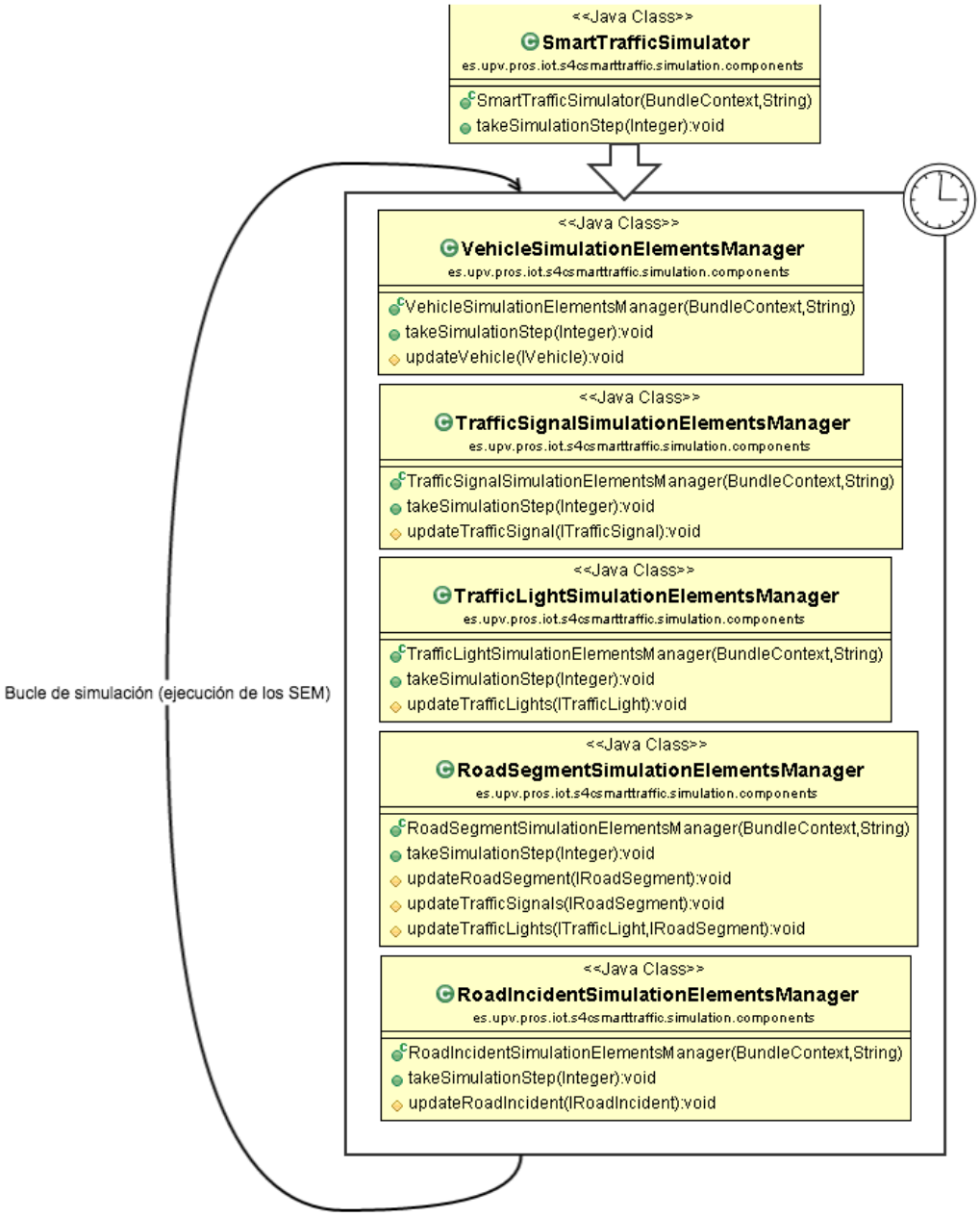

Figura 32: Diagrama de clases de los SimulationElementManagers

- **Activator**

Implementa el método starter del sistema, y es donde se inicializan todos los componentes que formarán parte de él. Al iniciar el sistema, este activador es el que se ejecuta y comienza a cargar e iniciar todos los componentes:

  - Se establece el contexto OSGi.
  - Se carga la parametrización de simulación y de los configuradores
  - Se pasa a cargar e instanciar el mapa de carreteras junto a las señalizaciones, los semáforos y los vehículos.
  - Se definen las rutas a los navegadores de los vehículos y se activan.
  - Se instancian los elementos SEM (Simulation Element Managers) descritos anteriormente.
  - Empieza la simulación con los parámetros indicados

```
public void start(final BundleContext bundleContext) throws Exception {
	Activator.context = bundleContext;
	final String simulatorID = "simulator";
	// SMART THINGS
	// ** MAP ** Inicialización de 4 segmentos de carreteras
	RoadSegment s1 = new RoadSegment(Activator.context, "S1", 60, 10, 40);
	RoadSegment s2 = new RoadSegment(Activator.context, "S2", 70, 5, 50);
	RoadSegment s3 = new RoadSegment(Activator.context, "S3", 50, 10, 60);
	RoadSegment s4 = new RoadSegment(Activator.context, "S4", 40, 10, 70);

	s1.start();
	s2.start();
	s3.start();
	s4.start();

	// ** TRAFFIC SIGNALS ** Inicialización de 3 semáforos en 2 grupos
	final TrafficLightsController itl = new TrafficLightsController(bundleContext, "itl");

	ITrafficLight tl1 = new TrafficLight(bundleContext, "tl1");
	tl1.setLightsConfiguration("LH-").getTrafficLightConfigurator().setManagedByTrafficLightController(itl)
		.setRoadSegment(s1.getId()).setStartingPosition(5).setEndingPosition(5).start();
	ITrafficLight tl2 = new TrafficLight(bundleContext, "tl2");
	tl2.setLightsConfiguration("HL-").getTrafficLightConfigurator().setManagedByTrafficLightController(itl)
		.setRoadSegment(s2.getId()).setStartingPosition(30).setEndingPosition(30).start();
	ITrafficLight tl3 = new TrafficLight(bundleContext, "tl3");
	tl3.setLightsConfiguration("HL-").getTrafficLightConfigurator().setManagedByTrafficLightController(itl)
		.setRoadSegment(s3.getId()).setStartingPosition(1).setEndingPosition(1).start();

	itl.addTrafficLightToGroup1(tl1);
	itl.addTrafficLightToGroup2(tl2);
	itl.addTrafficLightToGroup1(tl3);
	itl.start();

	// ** ROAD INCIDENT ** Inicialización de 2 incidentes
	new RoadIncident(bundleContext, "ri1", s1.getId(), "working-area", 10, 15).start();
	new RoadIncident(bundleContext, "ri2", s1.getId(), "crash", 30, 30).start();
	// ** /MAP **

	// ** VEHICLES ** Inicialización de 3 vehiculos y de una ruta compartida
	VehicleRoute r = new VehicleRoute("def");
	r.addRouteFragment(s1, 0, s1, 50);
	r.addRouteFragment(s2, 10, s2, 40);
	r.addRouteFragment(s3, 40, s3, 0);
	r.addRouteFragment(s4, 0, s4, 20);
	Vehicle c1 = new Vehicle(bundleContext, "c1",2, r);
	Vehicle c2 = new Vehicle(bundleContext, "c2",4, r);
	Vehicle c3 = new Vehicle(bundleContext, "c3",6, r);

	c1.start();
	c2.start();
	c3.start();
	// ** /VEHICLES **

	// SIMULATION Inicialización de los SEM
	new RoadSegmentSimulationElementsManager(bundleContext, simulatorID);
	new RoadIncidentSimulationElementsManager(bundleContext, simulatorID);
	new TrafficSignalSimulationElementsManager(bundleContext, simulatorID);
	new TrafficLightSimulationElementsManager(bundleContext, simulatorID);
	new VehicleSimulationElementsManager(bundleContext, simulatorID);

	final ISimulator sim = new SmartTrafficSimulator(bundleContext, simulatorID);
	sim.getConfigurator()
		.setSimulatorStepsManager(new TimedSimulatorStepsManager(sim).setSimulationTimeSteps(1000));
	sim.getSimulatorLifeCycleManager().start();
	// /SIMULATION
```

Figura 33: Activador de SmartTrafficSimulator

6.7. Monitorización (S4CSmartMonitoring)

La capa de monitorización incluye el despliegue de una serie de monitores que vigilan los valores de ciertos atributos de los componentes en el sistema de simulación. Está compuesta por 3 componentes principales; el comunicador, que incluye las clases de comunicación de tipo publicador y subscriptor, que conectan con el sistema de tráfico y con la capa de integración de usuario, el objeto monitor, definido como una clase abstracta con la estructura genérica de todas las implementaciones ad-hoc, y el activador donde se integra en el contexto de OSGi como un *bundle* junto a los demás componentes del sistema y se inicializan todas las instancias de monitorización.

- **MonitorCommunicator** (Sub y Pub)
  Las comunicaciones del módulo de monitorización son de tipo subscripción al sistema de gestión del tráfico, de donde obtiene la información, y de tipo publicador con el módulo de integración del usuario a quien retransmite una notificación al cumplirse las condiciones programadas en la instancia del monitor. La API de comunicaciones desarrollada es la siguiente:

  - monitorTopic(String topic): Se subscribe al canal indicado en el servidor, dicho canal es donde el gestor de tráfico publica los diferentes eventos.

  - sendNotification(final String rol, final String message): Publica al canal subscrito por el módulo de integración del usuario el mensaje para que este lo distribuya a un grupo de usuarios definido por su rol.

  - sendFeedbackRequest(final String rol, final String user, final String message, final String options): Mismo funcionamiento que sendNotification pero dirigido a un usuario individual y con un mensaje de tipo feedback, indicando el texto del mensaje y las opciones que se le ofrecen al usuario.

- **Monitor**

Se trata de una clase abstracta que implementa todo lo necesario para desplegar un monitor que escuche y retransmita diferentes temas o "*topics*" con un enfoque publicador/subscriptor. Las clases hijas deberán implementar el método *checkIncomingEvent* que es el que realiza tanto el filtro como el envío vía el publicador.

Pueden tratarse de monitores globales, que vigilen el contexto y busquen una visión completa del sistema, o pueden ser específicos a un elemento concreto, proviniendo una visión acotada a dicho elemento.

Sus atributos más destacados son:

- User y Rol: indican los destinatarios del aviso.
- Message: Texto del cuerpo de la notificación.
- listenerTopic y receiverTopic: Indican las direcciones de los canales de subscripción y publicación utilizados por el componente comunicador. Ambos atributos se almacenan dentro del OSGibean para poder acceder a sus valores vía el contexto.

Para el escenario propuesto se han implementado dos monitores, uno enfocado a una instancia **específica**, que notifica y solicita feedback al conductor del vehículo asociado ante un evento de tipo *roadIncident* en su ruta y otro **general** a cualquier instancia**,** que notifica al rol *traffic_operator* cuando una carretera cualquiera pasa a tener el estado "Congestionado".

- **RoadIncidentMonitor**
  El filtrado se basa en comprobar si es un mensaje de *infoType* "INCIDENT_TYPE". En caso afirmativo, se forma un mensaje de tipo **UserNotification** y se publica en el canal.

- **RouteCongestionMonitor**
  El filtrado se basa en comprobar si es un mensaje de *infoType* "TRAFFIC_CONGESTION" y si proviene de una carretera existente en la ruta del navegador del vehículo asociado. En caso afirmativo se forma un mensaje de tipo **UserMessageFeedback** indicando las opciones "Aceptar", "Cancelar" y "Otros" y se publica en el canal. A diferencia del anterior monitor, este es necesario ligarlo a un vehículo determinado para así poder obtener la información de su navegación.

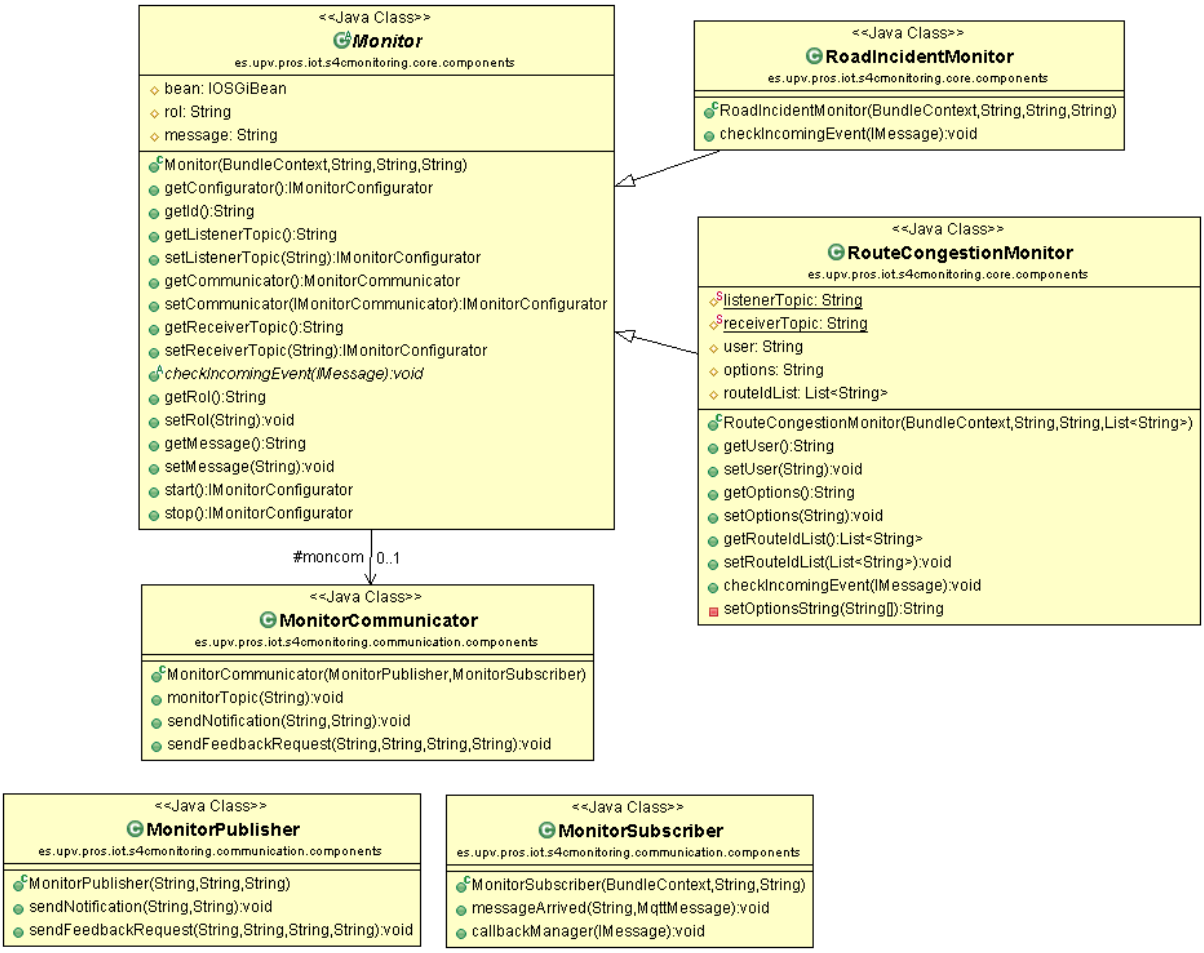

Figura 34: Diagrama de clases de la monitorización

- **Activator** (OSGi BundleActivator)

Al activar el *bundle* en el contexto de OSGi se instancian en el método "*Start*" todos los monitores que se deseen incluir en la capa de monitorización y se asocia con el contexto global del sistema OSGi.

```
@Override
public void start(final BundleContext bundleContext) throws Exception {
   Activator.context = bundleContext;

   // Creamos un monitor que escucha de todas las RoadSegments, canal info, filtra solo los incidentes
   // y transmite al s4ui por el canal notificaciones
   new RoadIncidentMonitor(context, "m1","/road/+/info/","/s4cui/requests/").start();

   // Creamos otro monitor que indicará si hay congestión en alguna RS de la ruta indicada
   new RouteCongestionMonitor(context, "m2", "c1", Arrays.asList("S1", "S2", "S3", "S4")).start();

   System.out.println("Monitoring started...");

}
```

Figura 35: Ejemplo de instanciación de los monitores

6.8. Integración del usuario (S4CUserIntegration)

El módulo **S4CUserIntegration** es el encargado de implementar la capa de interacción con el usuario y proveer así, una interfaz de comunicaciones entre el conjunto del sistema y los clientes de usuario. Se trata de un módulo independiente dentro del contexto OSGi (como *bundle*) y conectado con la monitorización y con los clientes de los usuarios finales. Su funcionalidad es la de recibir los eventos disparados por el módulo de monitorización y distribuirlos a sus destinatarios.

Las comunicaciones se realizan mediante la implementación de una API de comunicaciones subscrita a un canal de subscripción en el bróker MQTT. Dichos canales se definen dentro del contexto de OSGi compartido para compartir la información con los otros módulos.

El módulo está compuesto por 3 componentes, el comunicador, que incluye la API de comunicaciones con el servidor MQTT (sub y pub), la clase *UserIntegration*, que define los atributos y métodos necesarios para gestionar los mensajes de la monitorización hacia los clientes de usuario y el activador, donde se integra y se inicializa en el contexto de OSGi como un *bundle* junto a los demás componentes.

- **UserIntegrationCommunicator** (Sub y Pub)
  Las comunicaciones son de tipo subscripción, con la monitorización, y de tipo publicador con los clientes de usuario mediante el servidor MQTT, quienes informan al usuario final. La API de comunicaciones desarrollada es la siguiente:

  - monitorTopic(String topic): Se subscribe al canal indicado en el servidor. Dicho canal es donde el gestor de tráfico publica los diferentes eventos.

  - sendToUser(final String user, final IMessage message): Publica el mensaje al canal subscrito definido para los usuarios instanciándolo con el usuario indicado.

  - sendToRol(final String rol, final IMessage message): Mismo funcionamiento que sendToUser pero con el canal de rol.

- **UserIntegration**

Clase que implementa los atributos y métodos para poder realizar las retrasmisiones de mensajes entre el sistema y los clientes de usuario.

Los atributos más relevantes son:

- User y Rol: indican los destinatarios del aviso.
- listenerTopic: Canal subscrito donde espera los mensajes entrantes.
- sendingTopic: Canal publicador donde los clientes de usuario se subscriben para recibir la información. Es dinámico y varía en función del destinatario.

El funcionamiento es similar al módulo de monitorización, pero sin realizar ningún filtrado, simplemente debe recibir un mensaje por el canal de subscripción e instanciar el usuario final a quien está dirigido y realizar el envío mediante una publicación al canal instanciado. Dicho funcionamiento se realiza mediante un método *callback*, denominado *dispatcher*, dentro de la clase. Este se activa cada vez que la monitorización publica un mensaje, entonces realiza la instanciación mediante una sustitución del patrón con los valores de usuario y/o rol y procede a la retransmisión del mensaje a dicho canal.

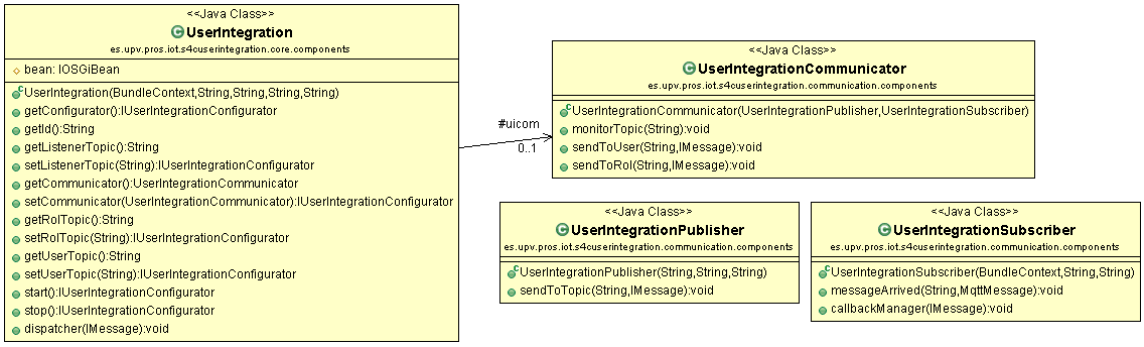

Figura 36: Diagrama de clases de UserIntegration

- **Activator** (OSGi BundleActivator)

Al activar el *bundle*, este se instancia en el método “Start” y se asocia con el contexto global del sistema OSGi.

```
public void start(final BundleContext bundleContext) throws Exception {
   Activator.context = bundleContext;

   new UserIntegration(context, "ui", "/s4cui/requests/", "/s4cui/rol/{id-rol}/","/s4cui/user/{id-rol}/{id-user}/").start();

}
```

Figura 37: Instanciación del módulo UserIntegration en el contexto OSGi

6.9. Cliente de usuario (Android App)

El cliente de usuario, que permite a los usuarios interactuar con el sistema, es desarrollado como una aplicación nativa Android para dispositivos superiores a la versión 4.0 Kit Kat. Las funcionalidades principales son las de ofrecer notificaciones al usuario y permitirles responderlas si así se les solicita.

La estructura de la implementación por lo tanto está ligada a los patrones de diseño propios de las aplicaciones Android y utiliza las herramientas y librerías propias del sistema. Ésta está compuesta principalmente de tres actividades principales y un servicio en segundo plano, o *fragment*, que se describen a continuación.

- **Fragment "Comunicaciones MQTT":** El cliente tiene que conectarse con el servidor de comunicaciones para recibir y contestar las notificaciones que éste envíe. Para ésta implementación se ha elegido un servidor de comunicaciones subscriptor, publicador con la tecnología MQTT. Para que el cliente Android se conecte al servidor, éste tiene que iniciar un servicio en segundo plano estableciendo la conexión y gestionarla, además de notificar a la llegada de un mensaje y de enviar sus respuestas. Todas estas funcionalidades están implementadas mediante un objeto de tipo *fragment* que se inicia en la actividad de *login* y permanece activo hasta el cierre de sesión. Para ello, se ha utilizado la misma librería externa que en el sistema para gestionar las comunicaciones MQTT y los mismos formatos de mensaje orientados al usuario, los "*UserMessage*", basados en JSON.

  El funcionamiento del fragmento es el siguiente:

  - Conectarse con el servidor MQTT y establecer la conexión.
  - Establecer un *callback* para cuando se reciba un mensaje por un canal de subscripción.
  - Enviar a un canal de publicación la respuesta a una petición de feedback.

Todos los parámetros necesarios se encuentran almacenados en la configuración de la aplicación mediante **SharedPreferences**. Esto permite almacenar persistentemente asociaciones (atributo, valor) sin necesidad de disponer de una base de datos. Al recibir un mensaje, se activa un método *callback* que lee el mensaje y lo convierte en un mensaje de tipo **UserMessageNotification** o **UserMessageFeedback** y solicita al sistema Android que lance un evento de tipo **Notificator** con la información proveniente del mensaje. Este evento es tratado por el sistema Android creando una notificación de tipo *pop-up* que avisará al usuario de la actividad de la aplicación. Si el mensaje era de solicitud de feedback, en el propio pop-up el usuario podrá seleccionar entre las opciones indicadas, que venían como atributos en el propio mensaje, y la respuesta es enviada al canal de publicación definido. Como funcionalidad final, si el usuario no está con el cliente en primer plano en su dispositivo, la selección de la notificación tiene como respuesta la recuperación de la aplicación a primer plano en la actividad de "Listado Principal".

- **Actividad "Identificación del usuario"**: Corresponde con la implementación de la funcionalidad que permite al usuario indicar su identificador, su rol y una contraseña de seguridad que le permite conectar con el servidor de comunicaciones MQTT y subscribirse al canal que esté interesado.

  La actividad cumple con el patrón de diseño y el *layout* **material design** definido por **Google** para las actividades de tipo "*LoginActivity*". Al verificar credenciales, lanza en segundo plano el *mqttFragment* para realizar y mantener la conexión con el servidor. Si la conexión es satisfactoria llama a ejecución a *MainActivity* pasándole la referencia del *fragment*.

- **Actividad "Listado Principal":** La funcionalidad principal es la de mostrar el listado de notificaciones recibidas desde el inicio de la aplicación. Estos datos solo se guardan en memoria, por lo que no son persistentes. El layout de la actividad está formado por varias partes. Por un lado, el área de notificaciones, implementada mediante un *RecyclerView* que permite la inserción de fragmentos de información denominados tarjetas, o *cards* en tiempo de ejecución. Estas tarjetas están formadas por la **información textual** de la notificación, su **título**, el ***timestamp*** de recepción y las **opciones de respuesta** si las hubiera en caso de *feedback*. Además, está formado por un menú desplegable propio de *material design* que permite al usuario ver su información de conexión y acceder a la actividad de configuración para cambiar algún parámetro.

- **Actividad "Gestión de Parámetros":** La funcionalidad principal de esta actividad es la de gestionar todos los parámetros que la aplicación cliente necesita para efectuar las conexiones y las comunicaciones con el servidor. Estos parámetros son almacenados en la configuración de la aplicación mediante **SharedPreferences**. Los parámetros más importantes son:

  - **Broker_URL**: Dirección URL del servidor MQTT.
  - **Username**: Identificador del cliente con el servidor MQTT.
  - **Pass**: Contraseña del cliente con el servidor MQTT.
  - **ClientID**: Identificador del usuario con nuestro sistema.
  - **Pub_topic**: Canal de publicación para los *feedbacks* recibidos.
  - **Sub_rolTopic**; Canal subscrito para las comunicaciones dirigidas a los diferentes roles.
  - **Sub_userTopic**; Canal subscrito para las comunicaciones dirigidas a los identificadores de usuario.

7. Prototipo funcional S4CSmartTraffic

7.1. Demostración

En el presente capítulo se describe el funcionamiento del sistema utilizando el escenario propuesto. Se describen, la configuración de entrada de los componentes, su comportamiento durante la ejecución del simulador y la intervención del humando en el sistema.

7.1.1. Descripción del escenario

El escenario está compuesto de los siguientes componentes con la siguiente configuración. Las posiciones se representan con el siguiente formato: **(*RoadSegment*, posición).**

- Circuito de **carreteras** (*RoadSegments*) unidas mediante varias intersecciones formando la siguiente formación. Todas con la siguiente configuración:
  - Posiciones: Indicadas en cada segmento.
  - Capacidad máxima: 2
  - Velocidad máxima: 10
- **Semáforo TL_RS6_5**
  - Estado: Luces en "Verde": Pasos de simulación pares.
  - Estado: Luces en "Rojo": Pasos de simulación impares.
  - Posición: (RS4,8)
- **Semáforo TL_RS8_10**
  - Estado: Luces en "Verde": Pasos de simulación impares.
  - Estado: Luces en "Rojo": Pasos de simulación pares.
  - Posición: (RS12,7)
- **Incidente de tráfico**
  - Tipo: Colisión de vehículos.
  - Afecta los pasos de ejecución: [5,7]
  - Posición: (RS16,1)
- **Vehículo 1**
  - **Navegador → Ruta 1**
  - Velocidad = 8
  - Estado = "*Stopped*"
  - Posición: (RS2,0)
- **Vehículo 2**
  - **Navegador → Ruta 2**
  - Velocidad = 10
  - Estado = "*Stopped*"
  - Posición: (RS8,0)

- **Ruta 1:** Ruta roja en la imagen.
- **Ruta 2:** Ruta morada en la imagen.
- **Monitor** de saturación del tráfico: Se activa cuando la carretera asociada pasa a estado Saturado y cuando deja de estarlo.
- **Monitor** de incidentes de tráfico: Se activa cuando un incidente aparece en las carreteras del circuito y cuando se solventan.
- Un **usuario** con el rol "TrafficManager".
- Un **usuario** con el rol "Vehicle Driver", representa el conductor del vehículo 1.

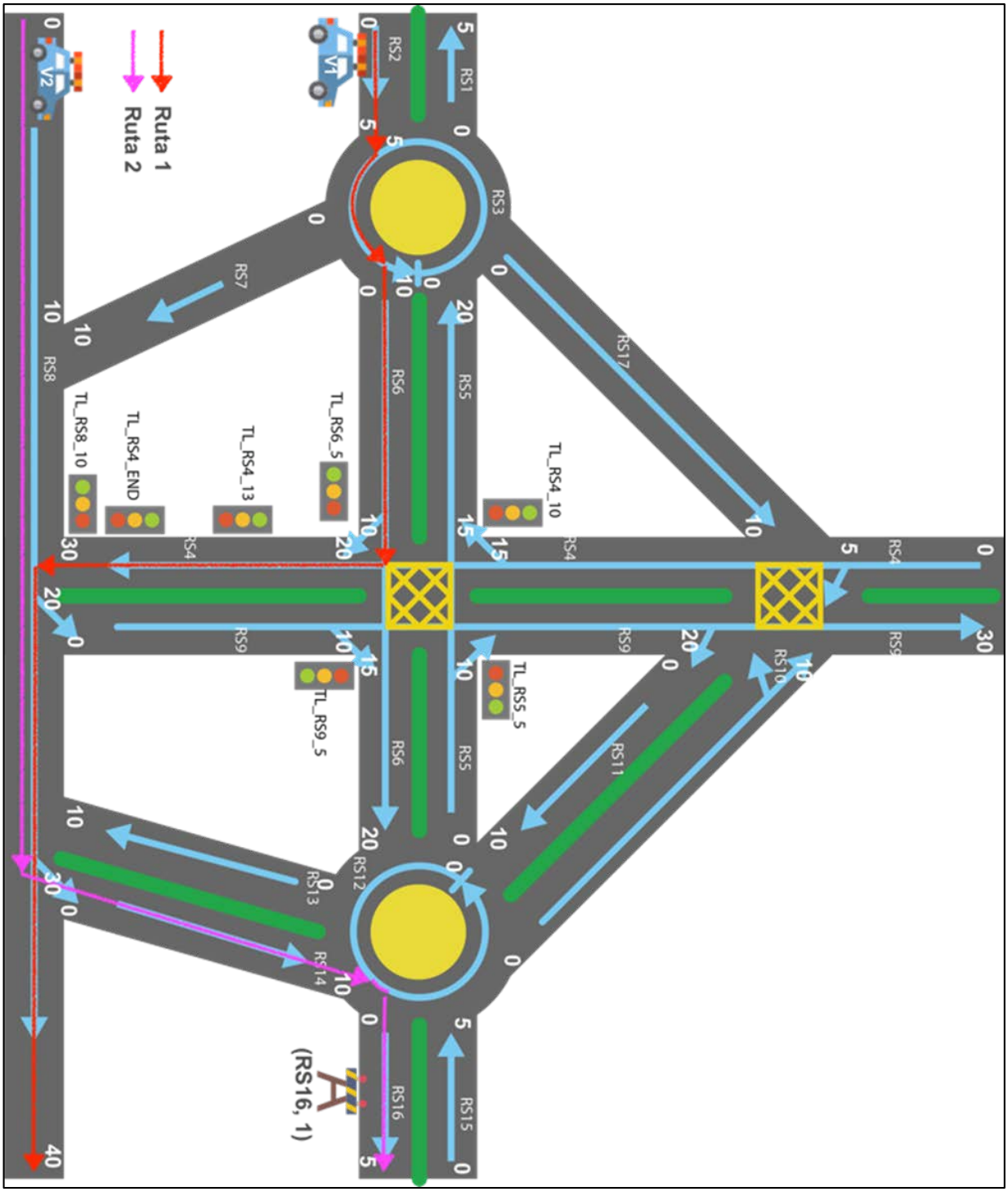

Figura 38: Mapa simplificado del escenario

7.1.2. Simulación del escenario

A continuación, se describe el comportamiento del sistema en cada paso de simulación. Para ello se ha utilizado el mapa de la anterior figura, "Mapa simplificado del escenario".

Instancia de objetos	Paso 1	Paso 2
Vehículo 1	(RS2,0) → (RS6,1)	(RS6,1) → (RS6, 9)
Vehículo 2	(RS8, 0) → (RS8, 10)	(RS8, 10) → (RS8, 18) "TL_RS8_10 impide el paso"
Semáforo TL_RS6_5	Rojo	Rojo → Verde
Semáforo TL_RS8_10	Verde	Verde → Rojo
Incidente I_RS16_1	(No activo)	(No activo)

Paso 3	Paso 4	Paso 5
(RS6,9) → (RS6, 9) "TL_RS6_5 impide el paso"	(RS6,9) → (RS4, 26)	(RS4,26) → (RS8, 23)
(RS8, 18) → (RS8, 28)	(RS8, 28) → (RS14, 7)	(RS14, 7) → (RS16, 0)
Verde → Rojo	Rojo → Verde	Verde → Rojo
Rojo → Verde	Verde → Rojo	Rojo → Verde
(No activo)	(No activo)	Activo

Paso 6	Paso 7	Paso 8
(RS8,23) → (RS8, 31)	(RS8, 31) → (RS8, 39)	(RS8, 39) → (RS8, 40) "Fin de ruta"
(RS16, 0) →(RS16, 0)	(RS16, 0) →(RS16, 0)	(RS16,0) → (RS16, 5) "Fin de ruta"
Rojo → Verde	Verde → Rojo	Rojo → Verde
Verde → Rojo	Rojo → Verde	Verde → Rojo
Activo	Activo	(No activo)

7.1.3. Interacción con el usuario o “Human in the Loop”

Independientemente del funcionamiento autónomo del sistema de tráfico, el usuario puede comunicarse con él mediante un cliente de usuario, para el escenario se ha desarrollado la aplicación Android ya descrita en el apartado de implementación. Para su uso el usuario se instala la aplicación en su dispositivo Android y sigue los siguientes pasos.

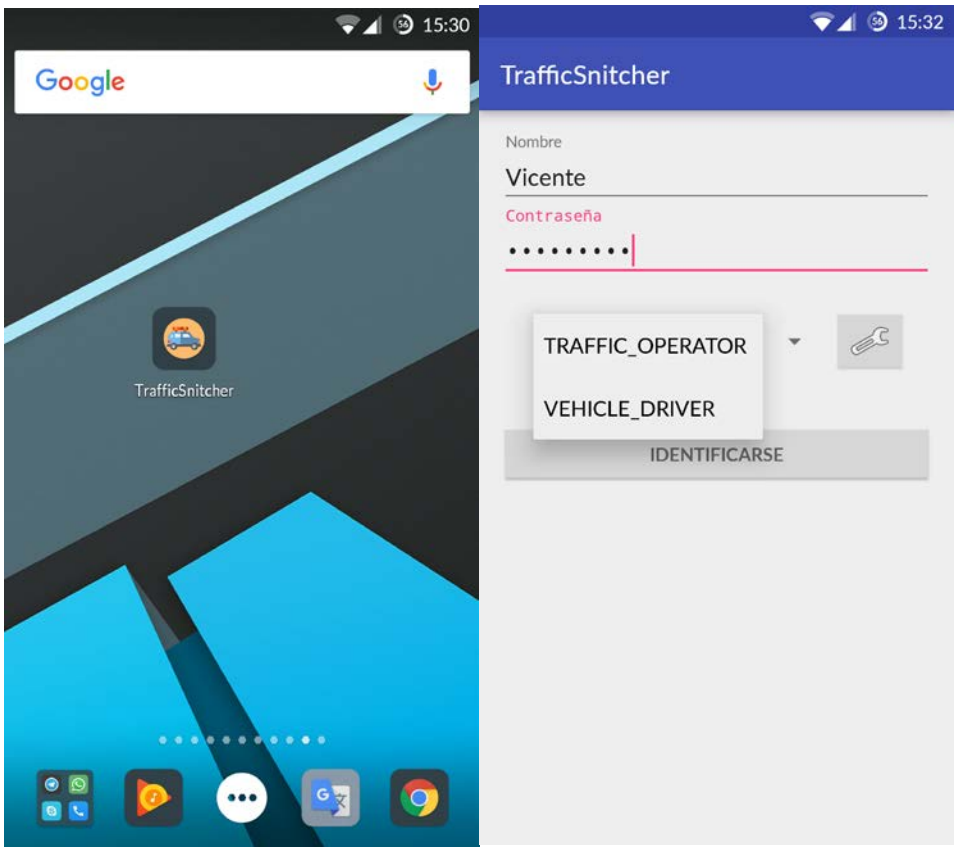

Figura 39: Icono de la aplicación y pantalla de identificación

Primero introduce sus datos de usuario en la pantalla de inicio de sesión.

Si es la primera vez que inicia el cliente puede que los datos de conexión por defecto no sean los correctos, por lo que puede entrar en la pantalla de configuración y personalizar los parámetros de conexión y de la aplicación y una vez modificados conectarse.

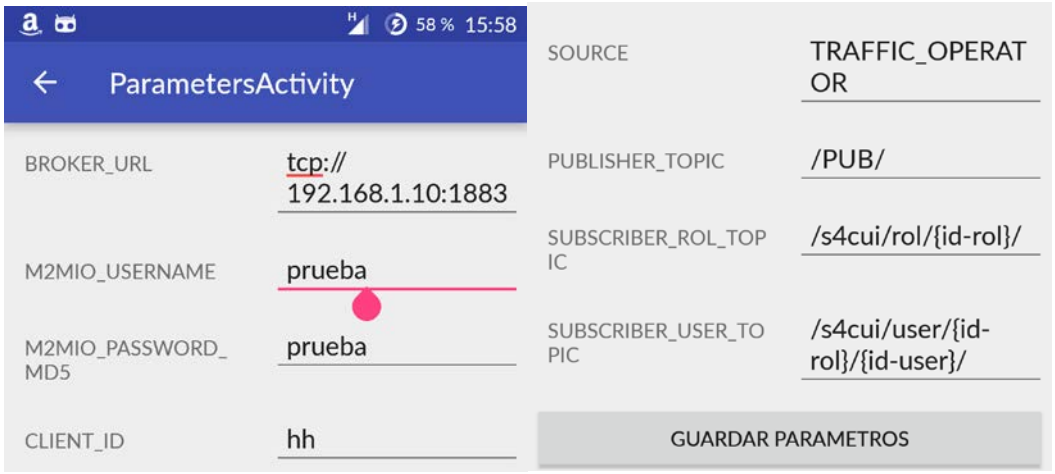

Figura 40: Pantalla de preferencias

Si los datos de usuario y de conexión son correctos, el usuario ya está a la espera de las notificaciones del servidor. Cuando se recibe una, ésta aparecerá como una notificación propia del sistema Android, haciendo vibrar el dispositivo y haciendo sonar el tono de notificaciones. Aun no estando la aplicación en primer plano, las notificaciones son recibidas y al pulsar en ella, el usuario es redirigido al listado de notificaciones recibidas. En él, el usuario puede consultar el historial de las notificaciones recibidas.

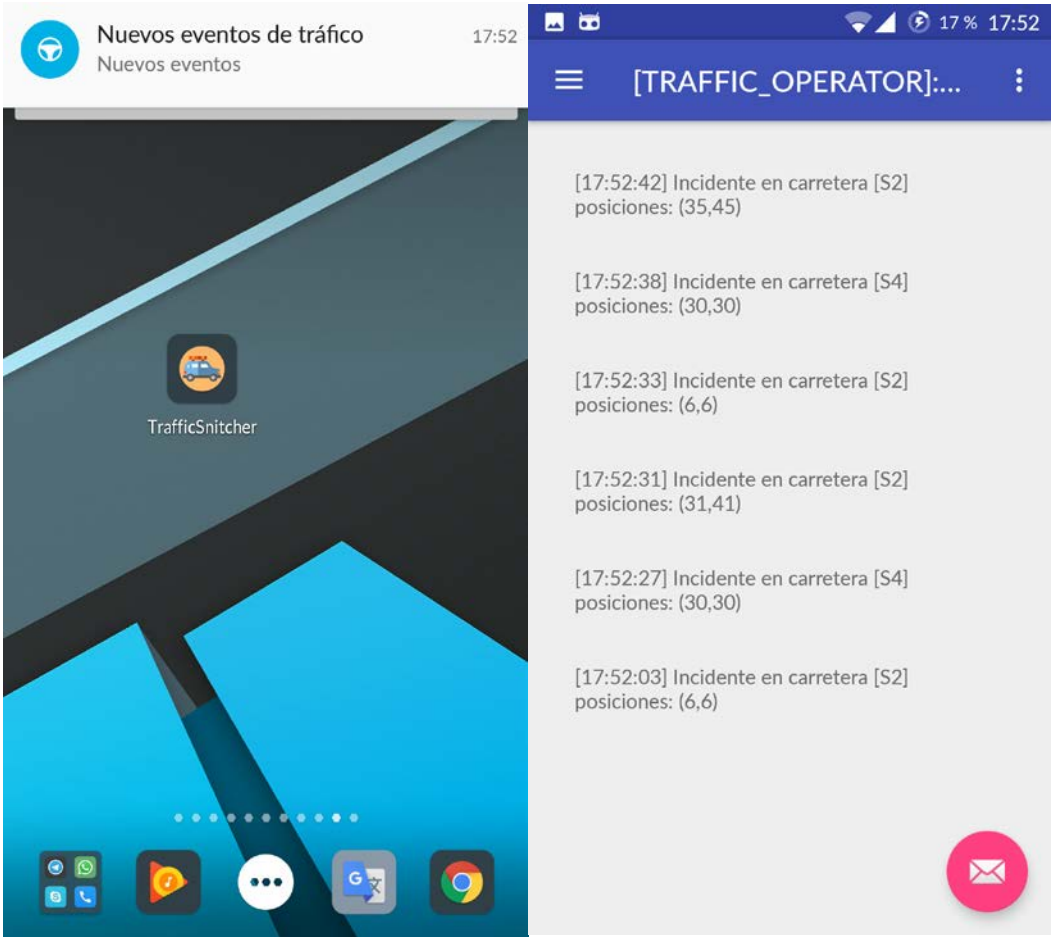

Figura 41: Notificación del sistema y listado principal

Si se tratan de peticiones de feedback, como en el caso de las notificaciones de saturación en carreteras dirigidos al rol "*vehicle driver*", la notificación del sistema Android estará formada además por una serie de opciones a las que el usuario puede responder, o también pueden omitirlas.

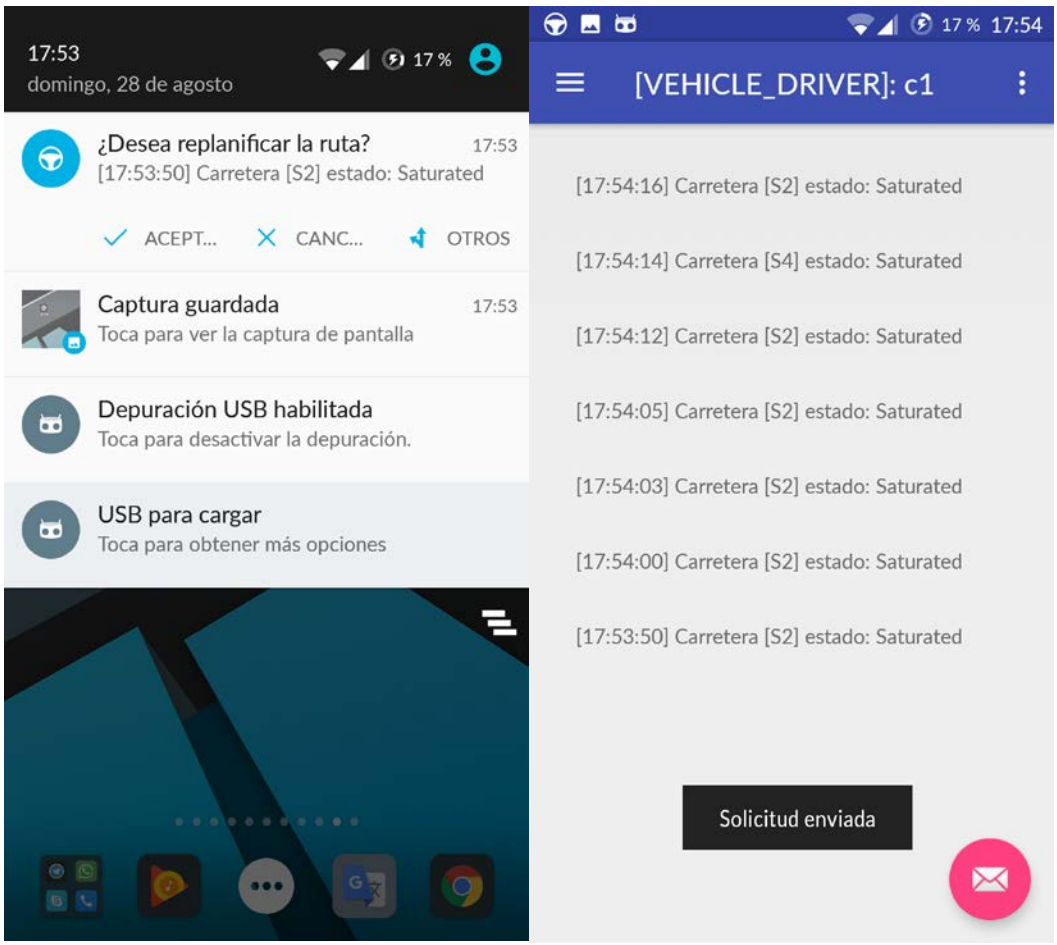

Figura 42: Notificación de feedback y notificación del envío

El usuario estará recibiendo mensajes del sistema dirigidos a su usuario y rol hasta que este decida cerrar sesión o pierda la conexión con el servidor. Para cerrar sesión solo tendrá que entrar en el menú contextual en el listado principal de la aplicación y seleccionar **desconectarse**.

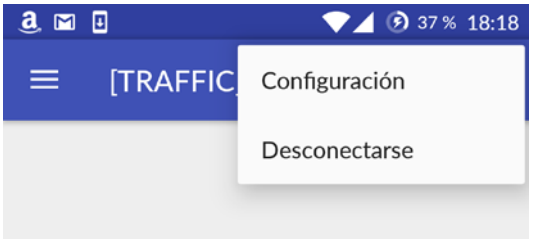

Figura 43: Menú de desconexión

7.2. Inclusión en escenarios reales

El escenario propuesto ha sido desarrollado dentro de un marco teórico mediante un simulador para poder diseñar, desarrollar y enseñar el funcionamiento del sistema de manera sencilla y sin tener que haber invertido mucho en equipamiento. Sin embargo, el objetivo final de este proyecto y de otros que le sucedan es de poderlo implementar en un escenario real con servicios públicos, dentro del marco de una ciudad inteligente que nos permita acceder a su información contextual, en nuestro caso por ejemplo, estados de los semáforos, incidentes de tráfico, velocidades máxima, etc.... y poder involucrar a los ciudadanos a participar en el sistema autónomo , como ya se ha visto, de manera pasiva mediante notificaciones de su interés, y de manera activa, pudiendo participar en la toma de decisiones o proveyendo feedback del sistema para su evolución.

A grandes rasgos sería necesaria, por una parte, la disposición pública de la información del entorno de la ciudad, proyectos como **VLCi** de la ciudad de valencia están proporcionando distintas categorías de información pública para los ciudadanos y con la que se podrían crear proyectos para la mejora de vida de los ciudadanos. Además, sería necesario involucrar a los propios usuarios, por ejemplo, los conductores y la dirección de tráfico de la ciudad, para que utilizaran el servicio mostrándoles las ventajas que podrían obtener y que proporcionaran feedback para la mejora continua del sistema.

Además, el presente proyecto provee de una completa base de la que pueden salir numerosos proyectos que implementen nuevos servicios o mejoren los presentes. En el capítulo de conclusiones se describen algunos proyectos futuros que quedaron fuera del alcance de este proyecto.

8. Conclusiones

8.1. Objetivos alcanzados

El resultado final de este proyecto es el de ofrecer un sistema de gestión de tráfico autónomo, implementado mediante un simulador, pero diseñado expresamente para poder ser desplegado en diferentes entornos tanto teóricos como reales, teniendo como referencia la plataforma IoT, VLCi. Además, otro objetivo principal era el de desarrollar toda la infraestructura necesaria para proporcionar un API de comunicaciones entre el sistema y los usuarios y que este forme parte activa en el funcionamiento del sistema. Todos estos objetivos han sido cumplidos con éxito e implantados mediante el escenario descrito en este documento.

Tanto el sistema como el cliente de usuario han sido implementados y a día de hoy están siendo utilizados para diferentes proyectos paralelos relacionados que expanden sus funcionalidades y lo implementan en diferentes entornos. Una pequeña descripción de estos se expone en la sección “Ampliaciones y proyectos futuros” del presente capítulo.

El presente documento está orientado como descripción de la arquitectura y componentes del sistema como documento de apoyo para el desarrollo de los futuros servicios y módulos que utilicen el proyecto.

8.2. Valoración personal

Personalmente, el desarrollo de este proyecto me ha servido para formar parte de un equipo de proyectos de investigación (I+D), ver de primera mano los procesos propios de estos desarrollos y poder aplicar conocimientos vistos dentro del master en ingeniería informática de la UPV. En especial considero como asignaturas más relevantes al desarrollo del proyecto la asignatura “Sistemas y aplicaciones distribuidas (SAD)”, “Inteligencia Ambiental (INA)” y “Planificación y dirección de Proyectos (PDP)”.

Además, me ha permitido descubrir e introducirme en un sector de la informática en auge actualmente como es el desarrollo de proyectos innovadores en inteligencia ambiental o IoT y de computación ubicua, además de ampliar mis conocimientos en patrones de diseño y programación modular y distribuida.

8.3. Ampliaciones y proyectos futuros

Este proyecto es la base para el desarrollo de otros que amplíen sus funcionalidades y permitan adaptarlo a diferentes escenarios. Algunos de los cuales iban a ser incluidos en el presente proyecto, pero tras evaluar el alcance del mismo se decidió no incluirlos.

- **Servicio de auto-adaptación:**

  El disponer de una arquitectura que permite separar al vehículo de su navegación nos permite poder cambiar sus atributos en tiempo real. Con esa premisa y la inclusión del humano dentro de las decisiones del sistema, se puede implementar un módulo que, a partir de las rutas definidas, el estado dinámico de las carreteras, el feedback del usuario y otras variables, proponga cambiar la ruta por una más óptima y que los usuarios sean los que decidan si cambiar o prefieren mantener la que ya tienen.

- **Parametrización del sistema mediante elementos gráficos:**

  Actualmente los parámetros de los componentes se introducen mediante parámetros o se incluyen en el código. Una ampliación podría ser el crear de una herramienta que, mediante elementos gráficos, el usuario pueda crear diferentes mapas de carreteras, personalizarlas y colocar los vehículos de manera más visual e intuitiva.

- **Interfaz visual del estado del simulador en tiempo real:**

  Siguiendo con la idea anterior, podría aprovecharse que todas las comunicaciones entre componentes pasan por un servidor de comunicaciones para monitorizarlos y con ayuda de la anterior herramienta visual, mostrar en una interfaz web el estado del simulador y sus componentes en cada paso del simulador, pudiendo incluso actualizar manualmente en vez de cada cierto periodo de tiempo. Esto sería muy útil para observar de manera gráfica que el comportamiento de los vehículos es el correcto e interactúa correctamente con el entorno, en vez de depurarlo mediante el uso de los log del sistema.

- **Ampliar funcionalidades del cliente de usuario**:

  Actualmente solo están implementadas las funcionalidades de notificar y mandar feedback del usuario, pero tanto el sistema como la arquitectura del cliente permiten incluir fácilmente otras funcionalidades que la aprovechen. Por ejemplo, podría ofrecer la interfaz visual del sistema a modo de visión global, muy útil para los gestores de tráfico, o podría poner en contacto a varios usuarios como si de una pequeña red social se tratara y notificarse entre ellos de eventos adicionales.

9. Bibliografía

[1] InndeaValencia. La Plataforma VLCi convierte a Valencia en la primera ciudad española cien por cien inteligente, 2015 [consulta 10-7-2015] Disponible en https://inndeavalencia.com/es/la-plataforma-vlci-convierte-a-valencia-en-la-primera-ciudad-espanola-cien-por-cien-inteligente

[2] FiWare, 2016 [consulta 6-9-2016] Disponible en https://www.fiware.org/

[3] Javier Cámara, Gabriel A. Moreno, and David Garlan. 2015. Reasoning about human participation in self-adaptive systems. In Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '15). IEEE Press, Piscataway, NJ, USA, 146-156.

[4] C. Evers, R. Kniewel, K. Geihs, L. Schmidt, The user in the loop: Enabling user participation for self-adaptive apps, Future Generation Computer Systems 34 (0) (2014) 110-123

[5] Stumpf, S., Burnett, M., Pipek, V. & Wong, W-K (2012). End-user interactions with intelligent and autonomous systems. In: J. A. Konstan, E. H. Chi & K. Höök (Eds.), CHI '12 Extended Abstracts on Human Factors in Computing Systems. (pp. 2755-2758). New York: ACM. ISBN 978-1-4503-1016-1

[6] ¿Qué es SCRUM? Proyectos ágiles.org, 2013 [consulta 10-8-2016] Disponible en https://proyectosagiles.org/que-es-scrum/

[7] Clúster ITC Audiovisual de Madrid. Internet de las cosas: Objetos interconectados y dispositivos inteligentes, 2015 [consulta 1-9-2016] Disponible en https://actualidad.madridnetwork.org/imgArticulos/Documentos/635294387380363206.pdf

[8] Caliper Corporation. Introducción al sistema TransModeler. 2015. [Consulta 3-9-2016] Disponible en: http://www.caliper.com/transmodeler/descripcion.htm

[9] Martin Treiber. TrafficSimulator.de, 2013. [Consulta 3-9-2016] Disponible en: http://www.traffic-simulation.de/

[10] J. Wuttke, Y. Brun, A. Gorla, and J. Ramaswamy. ADASIM and the automated traffic routing problem (ATRP), 2015. [Consulta 3-9-2016] Disponible en: https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/model-problem-atrp/

[11] Arquitectura de OSGi, 2016 [consulta 13-8-2016] Disponible en: https://www.osgi.org/developer/architecture/

[12] Joan Fons i Cors, Open Services Gateway initiative, Diseño e implementación de sistemas AmI. Master ISMFSI 2008/2009. 27p

[13] MQTT FAQ, 2015 [consulta 15-8-2016] Disponible en http://mqtt.org/faq

[14] Joan Fons i Cors, Seminario de INA: Comunicación Asíncrona con MQTT. Inteligencia Ambiental, Master MUIINF, 2015, 18p

[15] Introducción a JSON. 2012. [Consulta 1-9-2016]. Disponible en: http://www.json.org/json-es.html

[16] Guía de la API de Android Developer, 2016 [consulta 25-8-2016] Disponible en https://developer.android.com/guide/index.html

[17] IBM. Sistemas ciber-físicos y ciudades inteligentes. 2015 [consulta 4-9-2016] Disponible en http://www.ibm.com/developerworks/br/library/ba-cyber-physical-systems-and-smart-cities-iot/

[18] Oracle. RESTful web services. 2013 [Consulta 9-9-2016] Disponible en http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html

[19] ZeroMQ Guide, 2015 [consulta 15-8-2016] Disponible en http://zguide.zeromq.org/page:all