UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Máster en Ingeniería de Computadores

Universitat Politècnica de València

Adelantamientos seguros mediante un sistema de transmisión de vídeo entre vehículos

Trabajo Fin de Máster

Autor: *Javier Herrero Arnanz*

Tutor: *Carlos Tavares Calafate*

Septiembre de 2014

Agradecimientos

En primer lugar, dar las gracias a mi tutor, Carlos Tavares Calafate, por darme la oportunidad de llevar a cabo este proyecto, por todos sus consejos e ideas, y por escuchar mi opinión en todo momento.

Asimismo, agradecer al resto del profesorado todos los conocimientos aportados en cada una de las asignaturas del máster, pues me han sido de gran utilidad en la elaboración de este trabajo final.

Por último, dar las gracias a mi familia y a mi pareja por su apoyo incondicional en todo lo que hago, porque sin ellos no sería quien soy y nada de esto habría sido posible.

Resumen

El gran cambio que los dispositivos de telefonía móvil han experimentado en los últimos años los ha convertido en herramientas muy versátiles y potentes, capaces de proporcionar servicios muy diversos, algunos de ellos impensables hasta hace poco tiempo.

Aprovechando las nuevas características de estos dispositivos, en este trabajo se ha desarrollado un sistema para intentar contribuir a la mejora de la seguridad vial. En concreto, se ha implementado una aplicación para terminales con S.O. Android que asista a los conductores a la hora de realizar adelantamientos en situaciones de escasa visibilidad. Su cometido es el de establecer una conexión entre dos vehículos situados uno inmediatamente a continuación del otro, de tal manera que el vehículo situado delante transmita un vídeo en tiempo real que muestre lo que está visualizando al vehículo situado detrás de él.

Como puntos fuertes destacan: la implementación de un protocolo de comunicación C-S en el ámbito de las redes vehiculares ad-hoc (VANETs), la captura, transmisión y reproducción de vídeo en tiempo real, así como el uso de técnicas de geolocalización.

Índice de contenidos

1. Introducción

En las últimas décadas, uno de los sectores que más crecimiento y desarrollo ha experimentado ha sido, sin duda, el sector de las *Tecnologías de la Información y la Comunicación* (TIC). Los avances tecnológicos se encuentran presentes en la vida de casi cualquier persona. Desde el electrodoméstico más primario hasta el supercomputador más potente, todos ellos buscan mejorar algún aspecto de nuestras vidas (p.ej. salud, comunicación, entretenimiento, conocimiento, etc.), y es precisamente ahí donde radica su éxito.

Un claro ejemplo que pone este hecho de manifiesto es el caso de los dispositivos de telefonía móvil, los cuales han pasado de ofrecer unas pocas funcionalidades, como llamar o enviar mensajes de texto, a poseer grandes capacidades de cómputo y almacenamiento. Se han convertido en verdaderos computadores que caben dentro de nuestros bolsillos, una herramienta perfecta para ser aplicada en muy diversos ámbitos, ya no solo en el de la comunicación.

Este novedoso escenario ha despertado el interés de los desarrolladores por el uso de estos terminales inteligentes como instrumento para proporcionar servicios que no estaban cubiertos hasta el momento.

Dicho esto, este trabajo se centra en el desarrollo de una aplicación que sea capaz de mejorar, en cierto grado, un terreno que demanda una evolución continua: la seguridad vial. Para ello se hace uso de las nuevas capacidades de cómputo, de la infraestructura de red y de los sistemas de geolocalización de los que disponen estos dispositivos.

1.1 Objetivo

El objetivo principal de este proyecto consiste en el diseño e implementación de una aplicación para dispositivos móviles *Android*, que permita transmitir en tiempo real el vídeo capturado por la cámara del teléfono de un determinado usuario, desde su vehículo hasta otro situado inmediatamente detrás de él. De esta manera se pretende facilitar las operaciones de adelantamiento en aquellas situaciones en las que la visibilidad sea escasa.

Se pueden destacar dos componentes principales en el desarrollo del trabajo. El primero consiste en la implementación de un protocolo de comunicación cliente-servidor (C-S), que ha de ser capaz de establecer una sesión de streaming de vídeo únicamente entre dos dispositivos localizados en vehículos que circulan uno inmediatamente a continuación del otro, en la misma dirección y sentido. Todo ello

atendiendo a las características impuestas por el entorno en que se va a utilizar la aplicación, las *Redes Vehiculares Ad-Hoc* (VANETs por sus siglas en inglés).

Como segundo elemento principal destaca la transmisión y reproducción de vídeo en tiempo real, para lo cual se ha hecho uso del protocolo *RTSP*, pues para que el sistema sea capaz de asistir al conductor se debe garantizar la validez temporal y la calidad de las imágenes.

1.2 Motivación

La elección de las VANETs como objeto de estudio viene impulsada por el interés despertado hacia este tema en la asignatura "Redes inalámbricas ", impartida durante el máster.

Respecto a la posibilidad de enfocar el proyecto hacia una vertiente investigadora o profesional, se escogió esta última debido al atractivo que suponía la posibilidad de trabajar con dispositivos móviles inteligentes, pues se trata de un sector que ha crecido mucho en los últimos años, y en el cual existe una alta demanda de profesionales. Asimismo, destacar que desarrollar una aplicación para dispositivos móviles, permite enfrentarse a determinados problemas que posiblemente no aparecerían en un computador actual, como puede ser la falta de determinados recursos o la limitación de estos, lo que obliga a llevar a cabo un desarrollo más minucioso.

Por último, añadir que ya existen algunos trabajos que tratan esta materia, si bien es cierto que estamos ante un tema bastante novedoso. Hasta el momento se ha conseguido efectuar el intercambio de vídeo en tiempo real entre dos vehículos cercanos. Sin embargo, no se conoce ningún sistema automático capaz de seleccionar, de entre los distintos vehículos próximos, a aquel para el cual el vídeo es de utilidad, es decir, el situado inmediatamente detrás del vehículo emisor, tal y como se ha desarrollado en este trabajo.

1.3 Contenidos de la memoria

La presente memoria está organizada en un conjunto de secciones o capítulos, los cuales contienen información acerca de cada uno de los aspectos concretos que componen el proyecto. A continuación, se describe de forma breve de qué trata cada uno de ellos:

- **Capítulo 1 - Introducción:** Esta sección pretende situar al lector en el contexto en el que se desarrolla el proyecto, exponiendo la idea general en torno a la que gira, los objetivos que persigue, y las principales motivaciones que han impulsado su realización.

- **Capítulo 2 - Antecedentes:** En este apartado se presenta de manera general el concepto de redes vehiculares ad-hoc, y se realiza un breve repaso sobre otros trabajos desarrollados en el mismo ámbito.

- **Capítulo 3 - Entorno tecnológico:** Este epígrafe proporciona información relativa a aquellos elementos tecnológicos involucrados en el proyecto.

- **Capítulo 4 - Análisis del sistema:** En este capítulo se detalla el sistema propuesto, exponiendo los requisitos y necesidades que presenta.

- **Capítulo 5 - Diseño del sistema:** Este apartado describe con precisión la arquitectura del sistema y los distintos elementos que la componen.

- **Capítulo 6 - Implementación del sistema:** En esta sección se muestran los detalles de implementación de los principales elementos que componen el sistema.

- **Capítulo 7 - Pruebas:** En este epígrafe se describen las pruebas realizadas para validar el sistema y los resultados obtenidos.

- **Capítulo 8 - Conclusiones:** En este capítulo se realiza una comparativa de los resultados logrados frente a los objetivos marcados al comienzo, y se establecen las futuras líneas de trabajo.

- **Capítulo 9 - Bibliografía:** En la última sección se enumeran los recursos bibliográficos a los que se ha recurrido para la elaboración del trabajo.

2. Antecedentes

Una vez plasmada la idea en torno a la que gira el proyecto, es necesario establecer un punto de partida que permita conocer mejor el contexto en el que éste se engloba. Esto facilitará la lectura y el análisis del resto de capítulos, pues en ellos se profundiza en los aspectos más técnicos del trabajo.

Para tal propósito, en esta sección se realiza una breve descripción acerca de las VANETs, focalizando el interés en sus características principales. Asimismo, se mostrará cómo, a través de este tipo de redes, las TIC están comenzando a producir un cambio notorio en los sistemas de transporte actuales, volviéndolos más seguros y eficientes. Para finalizar, se expondrán otros trabajos realizados en el mismo ámbito.

2.1 VANETs: Vehicular Ad Hoc Networks

En la actualidad existe un automóvil por cada siete habitantes en el planeta, lo que hace un total de más de mil millones de automóviles en el mundo, convirtiéndolos en uno de los medios de transporte más populares. Su presencia en las carreteras crece día a día, a pesar de los accidentes, la contaminación y el consumo energético que suponen.

La eliminación de estos problemas no puede pasar por la sustitución de este medio de transporte en beneficio de uno más seguro, limpio y eficiente, sino que debemos transformar los automóviles para que sean capaces de adquirir dichas características.

Con este objetivo surgen las VANETs, un nuevo tipo de red de comunicación donde los nodos son los vehículos, y cuya utilidad radica en la provisión de una serie de nuevos servicios que se han denominado, de forma colectiva, *Sistemas Inteligentes de Transporte* (ITS por sus siglas en inglés). Es decir, las redes vehiculares ad-hoc proponen la aplicación de las TIC en el mundo de los transportes, en beneficio de la seguridad y la eficiencia de estos.

De esta manera, dotando a nuestros vehículos de la tecnología adecuada, seriamos capaces de conocer los accidentes que se producen en la vía con la suficiente antelación como para evitarlos, o elegir la ruta más óptima y eficiente en función de las condiciones de la carretera. Podríamos adecuar nuestro estilo de conducción para ahorrar carburante, o conocer las plazas de aparcamiento que hay disponibles en nuestra ciudad, incluso dispondríamos de conexión a Internet allá donde vayamos.

2.1.1 Paradigmas de comunicación en VANETs

Las VANETs son un tipo específico de redes MANET (Mobile Ad Hoc Network) pues, al igual que éstas, consisten en una red de dispositivos conectados inalámbricamente (vehículos), los cuales poseen propiedades de auto-configuración y gozan de cierta movilidad. Precisamente esa libertad de desplazamiento provoca que la topología de la red cambie de forma rápida y dinámica, hecho que condiciona de manera sustancial la forma en que se lleva a cabo la comunicación entre los nodos de la red.

En el caso concreto de las VANETs, distinguimos 3 paradigmas de comunicación principales [1] [2]: la comunicación entre vehículos (V2V), la comunicación entre vehículo e infraestructura (V2I, I2V), y la comunicación basada en técnicas de encaminamiento. Todas estas aproximaciones requieren de información muy precisa y actualizada acerca del entorno en el que se encuentran, lo que a su vez obliga a hacer uso de sistemas de posicionamiento muy exactos y de protocolos de comunicación inteligentes que permitan llevar a cabo el intercambio de la información. A continuación se describen brevemente estos tres tipos de comunicación, y se da un ejemplo de uso de cada uno de ellos:

- **Comunicación entre vehículos:** La comunicación entre vehículos (Figura. 2.1) utiliza mensajes broadcast/multicast multisalto para enviar información desde un vehículo origen a un grupo de vehículos destino, lo que significa que los vehículos que reciben el mensaje lo reenvían para que alcance a otros vehículos vecinos. El problema de esta aproximación se encuentra en que la red puede congestionarse a causa de tantos reenvíos de mensajes broadcast/multicast. Para mitigar este problema se opta por limitar los reenvíos en función de distintos parámetros, como son el instante de emisión, el número de saltos, o la localización del emisor. Este tipo de comunicación es útil en escenarios como el de un accidente, donde algunos de los vehículos implicados emiten un mensaje con el fin de alertar a los vehículos cercanos.

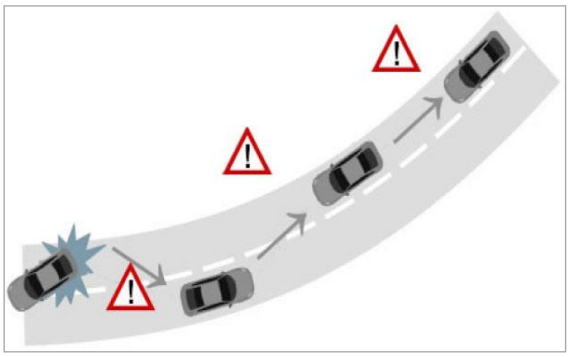


Figura 2.1 Comunicación entre vehículos.

- **Comunicación entre vehículo e infraestructura:** La comunicación entre vehículo e infraestructura (Figura. 2.2) consiste en una emisión broadcast de un solo salto desde un vehículo hacia una unidad en el borde de la calzada (RSU por sus siglas en inglés), o desde una RSU hacia todos los vehículos cercanos correctamente equipados. Por sus características, este tipo de comunicación proporciona un enlace con un gran ancho de banda entre los vehículos y las RSU, las cuales puede estar colocadas en intervalos iguales o menores a 1Km, con el fin de mantener altas tasas de transmisión de datos, incluso en condiciones de tráfico abundante. Un escenario donde tendría cabida esta aproximación consistiría en la emisión de alertas por parte de las RSU a aquellos vehículos que sobrepasen el límite de velocidad establecido para la vía en la que se encuentran.

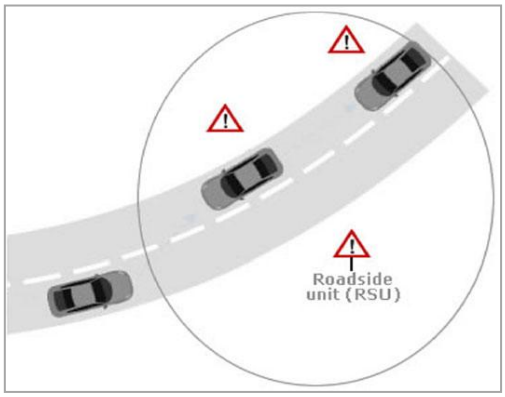


Figura 2.2 Comunicación entre vehículo e infraestructura.

- **Comunicación basada en encaminamiento:** La comunicación basada en encaminamiento (Figura. 2.3) emplea mensajes unicast multisalto que se propagan desde un vehículo/RSU origen hasta un vehículo/RSU destino a través de un conjunto de vehículos/RSUs intermedios cercanos. Para que el mensaje viaje correctamente desde el origen al destino, es necesario aplicar algún tipo de protocolo de encaminamiento, como por ejemplo DSR [3], AODV [4], OLSR [5], entre otros. Un ejemplo de uso de este tipo de comunicación sería el acceso a la WEB por parte de un vehículo a través de una RSU.

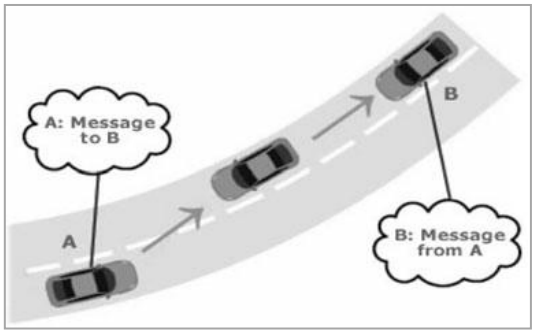


Figura 2.3 Comunicación basada en encaminamiento.

2.1.2 Estándar de comunicación inalámbrica IEEE 1609

Establecer conectividad inalámbrica entre los nodos de una VANET presenta muchos más desafíos que en las redes inalámbricas fijas convencionales, pues a estas últimas no les afectan factores como la variación de velocidad, los múltiples patrones de tráfico, y los distintos entornos de circulación existentes.

A causa de esto, se ha desarrollado una nueva tecnología de comunicación destinada al uso en entornos vehiculares, la cual se conoce como DSRC (del inglés, Dedicated Short Range Communications), y se integra en el estándar WAVE (del inglés, Wireless Access on Vehicular Environments). Esta tecnología está sujeta a múltiples normas emitidas por los distintos organismos de estandarización, siendo una de ellas la familia de estándares *IEEE 1609* [1] [2]. Esta familia define la arquitectura general, el modelo de comunicación, la estructura de gestión, y los mecanismos de seguridad y acceso físico para las comunicaciones inalámbricas en el entorno vehicular, todo ello distribuido en 4 capas bien diferenciadas (Figura. 2.4):

- **IEEE 1609.1:** Permite la interoperabilidad entre aplicaciones WAVE, describe los principales componentes de la arquitectura, y define los formatos de los mensajes de almacenamiento y comandos.

- **IEEE 1609.2:** Describe los servicios de seguridad de los mensajes de gestión y aplicación, con el fin de prevenir ataques como el espionaje, la suplantación de identidad, o la alteración de información.

- **IEEE 1609.3:** Recoge el direccionamiento y los servicios de enrutamiento dentro de un sistema WAVE para permitir el intercambio seguro de datos, y define el protocolo WSMP (WAVE Short Message Protocol) como alternativa a IP en aplicaciones WAVE.

- **IEEE 1609.4 - IEEE 802.11p:** Presenta los cambios realizados al estándar 802.11 para dar soporte a WAVE. Destaca el uso de canales de 10MHz en la banda de 5,9GHz (5.850-5.925 GHz), y comunicaciones con una tasa de transferencia que va desde los 3Mbps a los 27Mbps.

Asimismo, WAVE define dos tipos de dispositivos: las unidades en el borde de la calzada (RSU), y las unidades de a bordo (OBU), que esencialmente son dispositivos fijos y móviles, respectivamente. Tanto las RSUs como las OBUs pueden operar como proveedor o consumidor de servicios, y pueden cambiar entre estos dos roles. Sin embargo, lo habitual es que las RSUs alberguen aplicaciones encargadas de prestar servicios a otras aplicaciones cliente alojadas en las OBUs.

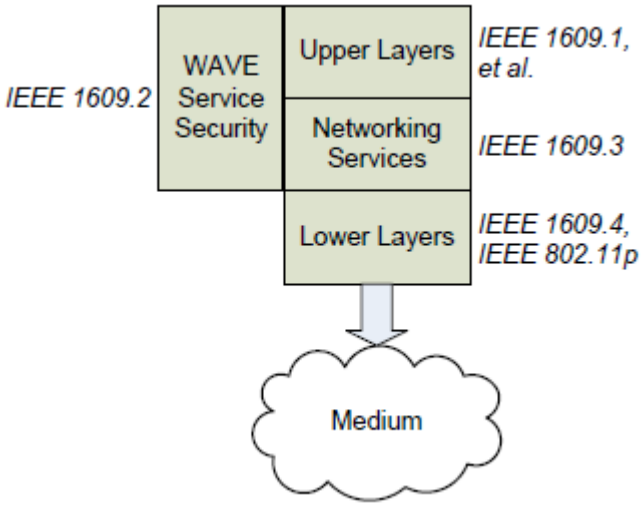


Figura 2.4 Arquitectura WAVE.

2.1.3 Principales desafíos

En los últimos años, las redes vehiculares ad-hoc han demostrado su utilidad e impacto en el mundo del transporte. No obstante, el desarrollo y futura expansión de esta tecnología pasa por superar los desafíos presentes en la actualidad.

El más importante de ellos tiene que ver con la seguridad pues, aunque como se ha expuesto anteriormente, la capa 1609.2 de la arquitectura WAVE se encarga de este cometido, nunca se puede garantizar la seguridad de un sistema al 100%. En concreto, los esfuerzos realizados en materia de seguridad deben ir encaminados a garantizar la privacidad de los conductores, la confidencialidad y autenticación de la información, la disponibilidad de los mecanismo de comunicación V2V y V2I, y la integridad de los datos transmitidos.

Otros retos pueden venir motivados por el medio físico empleado en este tipo de redes. Al ser el aire el canal utilizado para lograr las comunicaciones inalámbricas, surgen problemas como el desvanecimiento de las señales, la aparición de interferencias o el uso ineficiente del ancho de banda, cuyos efectos negativos hay que tratar de minimizar lo máximo posible.

Por último, cabe destacar la necesidad de aplicar esfuerzos no solo en el ámbito técnico, sino también en el socioeconómico. Sería de gran utilidad la implantación de mecanismos que permitan analizar y cuantificar el impacto de esta tecnología en los sistemas de transporte, para así ser capaces de establecer una relación coste-beneficio en cada nueva aplicación desarrollada.

2.2 Impacto de los ITS en la Seguridad Vial

Según el informe publicado por el *Grupo de colaboración de las Naciones Unidas* para la seguridad vial [6], 1,3 millones de personas mueren cada año en accidentes de tráfico, y entre 20 y 50 millones resultan heridas, principalmente en los países en desarrollo. Esto supone, en los sectores público y privado, pérdidas económicas de unos 518.000 millones USD en el mundo. Todas estas cifras hacen visible la necesidad, social y económica, de seguir invirtiendo en el desarrollo de nuevas aplicaciones dentro de los sistemas inteligentes de transporte (ITS).

En concreto, el estudio de las TIC en el ámbito del transporte, tuvo su comienzo a finales de la década de los 80 e inicios de los años 90. En un principio, el planteamiento de los trabajos elaborados en este terreno estaba enfocado fundamentalmente para tratar de mitigar la congestión de las vías y acortar los tiempos de traslado. No obstante, éstos impactaron de forma positiva en la seguridad vial, lo cual despertó el interés de muchos profesionales e investigadores que comenzaron a profundizar en este campo.

Hasta el momento, el impacto de los ITS ha sido bastante exitoso en la mayoría de los casos, convirtiéndose así en un mercado puntero en la actualidad. Los ITS pueden ayudar a reducir el número de accidentes, su gravedad, y el tiempo invertido por los servicios de emergencia en atender a las partes implicadas. En el campo de la seguridad vial, las aplicaciones ITS más significativas se pueden agrupar en cuatro grandes bloques [7]:

- **Sistemas de gestión y control de tráfico:** Estos sistemas intentan garantizar el máximo uso de las capacidades ofrecidas por las redes viales, implantando servicios como la coordinación de señales de tráfico para minimizar las demoras y controlar los atascos, medición y análisis de parámetros para mantener la densidad vehicular por debajo del nivel de saturación, detección y gestión de incidentes, accidentes y averías, entre otros. De esta forma, se logra reducir el número de accidentes, la contaminación y la congestión de las vías, así como un ahorro de energía.

- **Sistemas de información y asistencia a los usuarios:** A través de estos sistemas se pretende eliminar la incertidumbre en la conducción, la cual viene ocasionada por la falta de información en determinadas circunstancias. Se apoyan en la idea de que cuanta más información sobre las condiciones del sistema esté a disposición de los usuarios, mejor ajustarán éstos su tiempo, sus rutas y el modo de circular, lo cual a su vez repercutirá de forma positiva en la eficiencia y seguridad del sistema. En este ámbito, encontramos aplicaciones como la asistencia en la navegación, los mapas de congestión vial, el reconocimiento de señales, etc.

- **Sistemas de control de vehículos:** Estos buscan aumentar la seguridad mediante el control de la conducción, ya sea a través de medios que proporcionen mejor información con respecto al entorno, o asistiendo al conductor en el manejo del propio vehículo. Incluyen servicios como frenos antibloqueo, control de tracción, airbags inteligentes, sistemas anticolisiones, detección de puntos ciegos, emisión de alertas por cambio de carril, monitorización de conductor y ocupantes, entre otros.

- **Sistemas de gestión de emergencias:** Con estos se busca garantizar la asistencia ante una determinada emergencia, reduciendo así los tiempos de respuesta y atención, los cuales son factores que influyen de forma significativa en las posibilidades de supervivencia de las víctimas de un accidente de tráfico. Los ITS para la gestión de emergencias incluyen sistemas para la notificación automática de emergencias, sistemas para la optimización y guía de rutas para vehículos de emergencia, y soporte en las actividades de socorro.

Para finalizar, indicar que los avances de los ITS, así como su éxito presente y futuro, se deben en gran medida a su eficiencia y capacidad para solucionar problemas que no se podrían resolver o llevar a cabo de otro modo, por la falta de otros mecanismos capacitados. Asimismo, se han visto favorecidos por la globalización de los medios de comunicación e información, y la reducción de su coste.

Ahora bien, ha de tenerse en cuenta que, para mejorar la seguridad vial, no basta con el desarrollo de estas tecnologías. Los ITS sólo sirven como mecanismo de ayuda, por lo que seguirá siendo necesaria la evolución de factores sociológicos (p.ej. la educación, la moral y la conciencia), así como la inversión en infraestructuras para mantener las carreteras en buen estado.

2.3 Trabajos relacionados

Para afianzar las ideas y conceptos presentados en las secciones anteriores, se exponen a continuación algunos proyectos y trabajos desarrollados en el ámbito de los ITS, la seguridad vial y los dispositivos móviles.

Dentro del marco de la Unión Europea nació en el año 2002 *eSafety*, una iniciativa conjunta entre la Comisión Europea y numerosas compañías interesadas en la seguridad vial dentro de los sistemas de transporte modernos. El objetivo de esta iniciativa es aumentar la seguridad vial mediante la implementación y el desarrollo de sistemas de seguridad basados en las TIC, así como servir de punto de encuentro entre los distintos proyectos surgidos en este ámbito dentro de la comunidad europea. A

continuación se describen brevemente algunos de los proyectos desplegados en el marco *eSafety* [8] (Tabla 2.1).

Proyecto	Periodo	Financiación	Descripción
SEVECOM	2006-2009	Unión Europea	Tiene como objetivo definir e implementar de forma completa la arquitectura de seguridad en las comunicaciones V2V, V2I y I2V dentro de las redes vehiculares; *http://www.sevecom.org*
COOPERS	2006-2010	Unión Europea	Pretende mejorar la seguridad vial mediante la transmisión directa y actualizada de información acerca del tráfico existente en un tramo de autopista, entre la infraestructura y los vehículos; *http://www.coopers-ip.eu*
SAFESPOT	2006-2010	Unión Europea	Su propósito es establecer una red de cooperación entre los vehículos y la infraestructura, recogiendo y compartiendo información a bordo de los vehículos y en los márgenes de la calzada, de tal manera que se mejore la percepción que los conductores poseen acerca del entorno que rodea al vehículo, siendo capaces detectar situaciones potencialmente peligrosas con suficiente antelación; *http://www.safespot-eu.org*
CVIS	2007-2011	Unión Europea	Tiene como fin diseñar, desarrollar y probar nuevas tecnologías que permitan a los vehículos comunicarse entre sí y con las infraestructuras circundantes, de manera que aumente la seguridad, la eficiencia, y se reduzca el impacto ambiental de los sistemas de transporte; *http://www.cvisproject.org*
HAVE-IT	2008-2011	Unión Europea	Busca incrementar la seguridad y la eficiencia en los sistemas de transporte a través de una mayor automatización de los vehículos, de tal manera que exista un equilibrio entre las tareas realizadas por el conductor y las llevadas a cabo de forma automática; *http://www.haveit-eu.org*

Tabla 2.1 Proyectos eSafety.

Por otro lado, en los últimos años se están desarrollado numerosas aplicaciones para dispositivos móviles destinadas a proporcionar servicios muy diversos en los ITS. Su auge se debe en gran medida a las capacidades de computo, almacenamiento y conectividad de que disponen, y a que permiten llegar de forma rápida al cliente final, siendo capaces de aplicar esos servicios en casi cualquier tipo de vehículo independientemente de su antigüedad. Seguidamente se recogen algunos ejemplos de este tipo de aplicaciones (Tabla 2.2).

Nombre	SO	Descripción
ODBDROID	Android	Se encarga de monitorizar el estado del vehículo a través de un dispositivo ODB-II con el fin de detectar accidentes. La aplicación reacciona enviando la información relativa al accidente a través de un e-mail o un SMS a distintos destinos predefinidos, seguido inmediatamente por una llamada automática a los servicios de emergencia. Es capaz de reaccionar a los eventos de accidentes en menos de 3 segundos [9].
Driving Styles	Android	Es capaz de generar una clasificación de los estilos de conducción que posee un determinado conductor a lo largo de una ruta seguida. Para ello analiza la información obtenida a través de un dispositivo ODB-II, utilizando técnicas de minería de datos y redes neuronales. El objetivo final es ayudar a los conductores a corregir los malos hábitos en su forma de conducir, al tiempo que ofrece consejos útiles para mejorar la economía del combustible [10].
Waze	Android, iOS, Windows Phone 8, BlackBerry OS, Symbian, Windows Mobile	Se trata principalmente de una red social enfocada en la conducción. Proporciona asistencia en la navegación a través de mapas y de la colaboración de los propios usuarios. Éstos pueden reportar información acerca de diferentes eventos ocurridos en la carretera, con el fin de ayudar al resto de conductores en la toma de decisiones. Eventos como accidentes, localización de controles policiales y radares, localización de estaciones de servicio, estado del tráfico y de la calzada, etc.; *https://www.waze.com*
DriveAssist	Android	Permite que el conductor visualice información acerca del estado del tráfico cercano a través de una vista en forma de mapa. También es capaz de emitir alertas ante eventos próximos, como por ejemplo avisar del estado en el que se encuentra la vía por la que se está circulando [11].
CarbonRecorder	Android	Permite conocer las emisiones diarias de CO_2 emitidas por un determinado vehículo y compartirlas a través de las redes sociales. Su principal objetivo es fomentar un estilo de conducción más eficiente, concienciar a los conductores de la importancia de reducir las emisiones, así como servir de plataforma para la obtención de información útil en los trabajos de investigación dentro de las redes vehiculares [12].
iOnRoad	Android, iOS	Su cometido es el de alertar al conductor ante posibles peligros situados frente al vehículo. Hace uso de la cámara del dispositivo y de los sensores GPS para calcular la velocidad del vehículo y la distancia hasta cualquier objeto situado delante. Si el vehículo se aproxima demasiado a un peligro se dispara una señal audiovisual que alerta de una posible colisión, lo que permite al conductor frenar a tiempo; *http://www.ionroad.com*

Tabla 2.2 Aplicaciones ITS para dispositivos móviles.

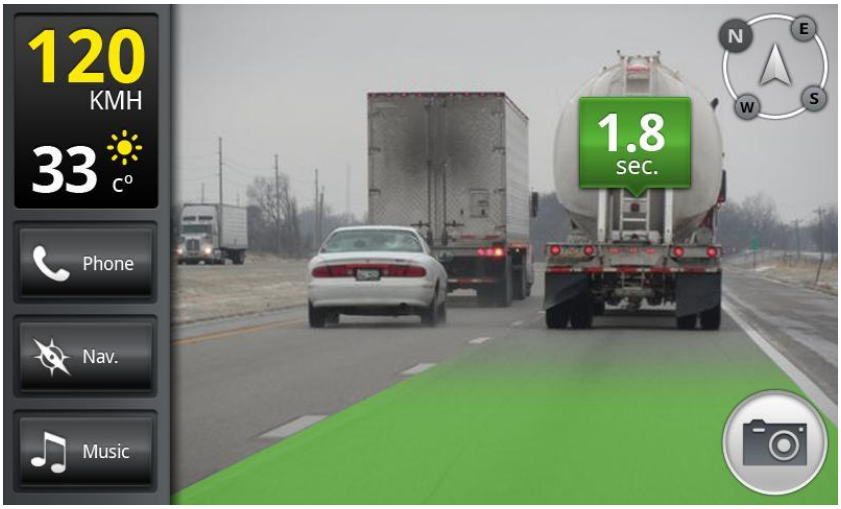


Figura 2.5 Captura de pantalla de la aplicación iOnRoad.

Por último cabe destacar un sistema que guarda muchas similitudes con el trabajo aquí presentado, pues aunque la implementación de ambos difiere en muchos puntos, el fin último es el mismo en los dos casos: asistir a los conductores en los adelantamientos a través de un vídeo en tiempo real que muestra la perspectiva del vehículo situado delante.

Su nombre es STS: *See Through System* [13], y su cometido es tratar de revertir la falta de visión experimentada por los conductores al realizar un adelantamiento. Esa falta de visión viene provocada por el vehículo situado justo delante, el que se desea adelantar, cuyas dimensiones en muchos casos convierten el adelantamiento en una maniobra realizada prácticamente a ciegas. Por lo tanto, lo que propone este trabajo es un sistema capaz de convertir los vehículos en objetos transparentes a través de los cuales se pueda visualizar el resto de la calzada. Esto ofrecerá mucha mas información al conductor de la que tenía antes, permitiéndole evaluar si la maniobra que va a realizar es o no segura y así terminar ejecutándola con éxito.

Para dotar a un vehículo de esta funcionalidad debe estar equipado con aquellos dispositivos que permitan establecer una comunicación con otros vehículos, así como capturar, transmitir y reproducir el streaming de vídeo. Por lo tanto, serán necesarios un ordenador portátil, un receptor GPS, un sistema de radio DSRC, una antena amplificadora, una cámara de alta resolución y un monitor LCD transparente. También es preciso un protocolo de comunicación a través del cual los vehículos puedan validar la utilidad del servicio y negociar sus condiciones. En líneas generales, en dicho protocolo se valida la distancia que separa al vehículo cliente del servidor, se examinan diferentes parámetros para asegurar que el adelantamiento se puede realizar con seguridad, y se negocia la resolución del vídeo en cuestión. Si el protocolo se completa satisfactoriamente el vehículo situado delante comenzara a emitir el streaming de vídeo hasta el vehículo situado detrás.

Llegados a este punto el vehículo cliente recibirá el vídeo, pero no lo reproducirá directamente, sino que construirá una perspectiva tubular con profundidad haciendo uso de las dimensiones del vehículo que le precede y de técnicas de realidad aumentada. De esta manera la visión que recibe el conductor del vehículo cliente es una perspectiva en tres dimensiones, la cual se asemeja de forma más realista a la vista que posee el vehículo servidor, y hace que éste parezca transparente de verdad.

En general, las pruebas realizadas sobre este sistema indican el buen comportamiento del servicio prestado. Los resultados muestran que el sistema satisface los requisitos de seguridad y latencia requeridos , así como la calidad del vídeo transmitido.

Como se verá más adelante, las principales diferencias existentes entre el sistema propuesto y STS se encuentran en el protocolo de comunicación, en los dispositivos utilizados y en el modo en que se reproduce el streaming de vídeo.

Figura 2.6 STS activado en una vía de dos carriles.

3. Entorno tecnológico

Antes de pasar al desarrollo del proyecto propiamente dicho, es necesario analizar y describir las diferentes tecnologías involucradas en ese proceso. En caso contrario sería imposible comprender muchas de las ideas tratadas en los capítulos siguientes.

En esta sección se estudiarán los cuatro elementos tecnológicos en los que se apoya el sistema, es decir: el sistema operativo sobre el que se desarrolla, la tecnología utilizada en las comunicaciones entre dispositivos, el sistema de geolocalización que permite obtener la ubicación de los vehículos, así como los protocolos empleados en la transmisión, recepción y reproducción del streaming de vídeo.

3.1 Android

Android es un sistema operativo basado en el kernel de *Linux*, el cual fue originalmente concebido para ser utilizado en dispositivos móviles con pantalla táctil, aunque en la actualidad también puede encontrarse en relojes, televisores y coches.

Android fue inicialmente desarrollado por *Android Inc.*, compañía fundada en Palo Alto (California, EE.UU) en Octubre del año 2003 por *Andy Rubin*, *Rich Miner*, *Nick Sears* and *Chris White*. Sin embargo, pasó prácticamente inadvertido hasta que *Google* lo compró en 2005, momento en el que empezaron a aparecer los primeros rumores acerca del nuevo sistema operativo.

El 5 de noviembre de 2007 se forma la *Open Handset Alliance*, un consorcio compuesto por varias compañías entre las que se encuentra *Google*, *Texas Instruments*, *Broadcom Corporation*, *Nvidia*, *Qualcomm*, *Samsung Electronics*, *Intel*, *LG*, *Motorola*, *T-Mobile*, entre otras, con el objetivo de desarrollar estándares abiertos para dispositivos móviles. Al mismo tiempo de su fundación, la OHA presentó la primera versión de Android (1.0 Apple Pie) como su primer producto, además del kit de desarrollo (SDK) para que los programadores empezaran a crear aplicaciones para este sistema.

Android permite desarrollar aplicaciones a través de una variante del lenguaje de programación Java, siendo necesarios tres componentes de software principales para este cometido:

1. El kit de desarrollo de Java estándar (Java Development Kit ó JDK).
2. El kit de desarrollo de Android (Android SDK).
3. Un entorno de desarrollo, como puede ser Eclipse o Android Studio.

3.1.1 Arquitectura Android

Android sigue una arquitectura basada en capas, donde cada una de ellas lleva a cabo un cometido específico y bien marcado [14]. Es importante saber que, en este tipo de arquitectura software, cada capa emplea los servicios proporcionados por la capa que se encuentra justo debajo, y por tanto, si se le da la vuelta al razonamiento, cada capa ofrece sus servicios a la capa situada encima. Esta distribución hace posible el acceso a las capas más bajas mediante el uso de librerías, de manera que el desarrollador puede utilizar los componentes hardware del dispositivo sin necesidad de realizar programación a bajo nivel. A continuación se muestra un diagrama con cada una de las capas y bloques que componen la arquitectura de este sistema operativo (Figura. 3.1).

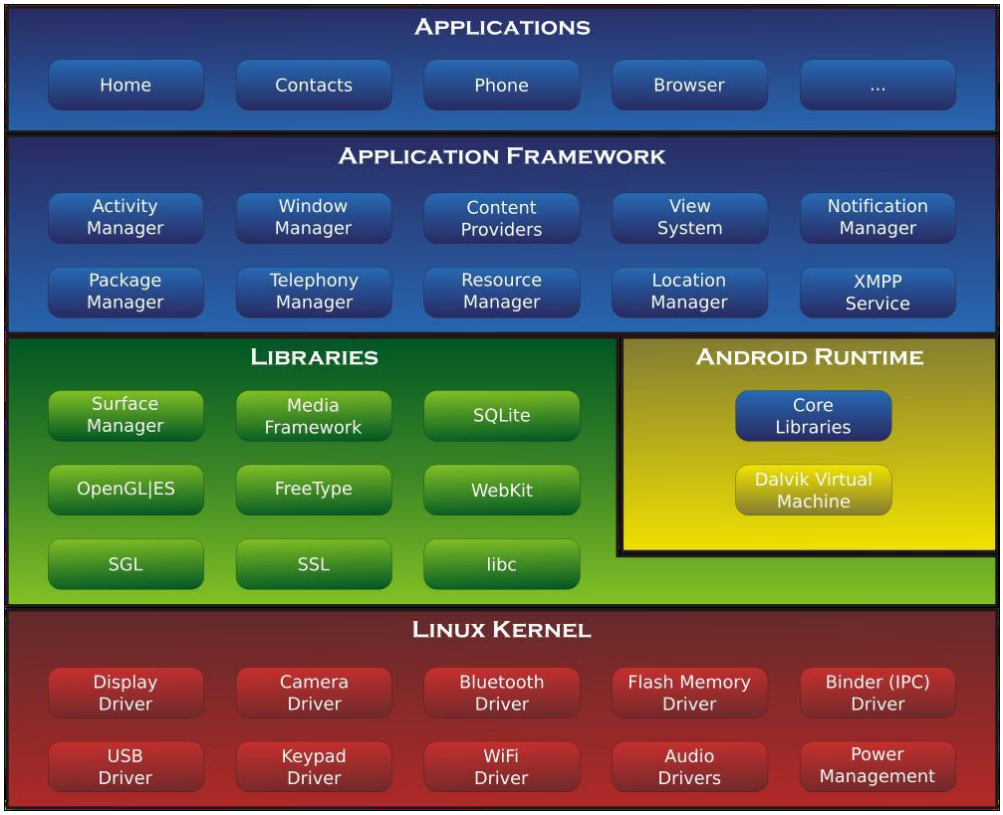


Figura 3.1 Arquitectura Android.

Kernel de Linux

Android establece su base y cimientos sobre una tecnología sólida y perfectamente probada: se trata del núcleo del sistema operativo Linux, creado por *Linus Torvalds* en 1991, y que actualmente puede encontrarse en todo tipo de ordenadores. Android incluye una versión modificada de este kernel, adaptada a las características hardware marcadas por el dispositivo móvil donde va a ejecutarse.

Este núcleo actúa como una capa de abstracción entre el hardware y el resto de capas de Android, lo que permite que este sistema operativo pueda portarse a una amplia variedad de plataformas.

El desarrollador nunca accederá directamente a esta capa, sino que empleará las librerías disponibles en las capas superiores. Para cada elemento hardware del teléfono existirá un controlador contenido en el kernel que permitirá usarlo desde el software.

Además de todo lo anterior, esta capa se encarga de la gestión de los recursos del dispositivo móvil, es decir, de la gestión de la memoria, de los procesos, llevar a cabo las operaciones de red y otros muchos servicios.

Librerías

La capa situada justo encima del kernel contiene las bibliotecas o librerías nativas de Android, las cuales están escritas en lenguaje C o C++ y son compiladas concretamente para la arquitectura hardware empleada en cada teléfono. Generalmente es el propio fabricante del dispositivo el que las crea, compila e instala. A través de estas librerías se van a proporcionar un conjunto de funcionalidades empleadas muy frecuentemente por las aplicaciones de más alto nivel. Desde la versión 1.5 existe la posibilidad de desarrollar bibliotecas nativas propias. Algunas de las más importantes son: *OpenGL* (motor gráfico 2D y 3D), *Bibliotecas multimedia* (reproducción y captura de audio, imagen y vídeo), *SSL* (cifrado de comunicaciones), y *SQLite* (bases de datos), entre otras.

Entorno de ejecución Android

Esta capa también se encuentra sobre el kernel y al mismo nivel que las bibliotecas nativas. Esto es así debido a que también está formada por librerías, por lo que no se le puede considerar como una capa independiente. En concreto, contiene las bibliotecas core de Java y la maquina virtual *Dalvik*, y será el lugar donde se lleve a cabo la ejecución de los procesos.

El componente principal del entorno de ejecución de Android es la máquina virtual *Dalvik*, diseñada y escrita por *Dan Bornstein*. Se trata de la implementación de la máquina virtual de Java por parte de Google, optimizada para dispositivos móviles. En general, todo el código Java escrito para Android se ejecutará en esta máquina virtual. *Dalvik* ejecuta archivos en formato .dex, los cuales se obtienen en tiempo de compilación a partir de los archivos clásicos de java, .class y .jar. La característica que hace especiales a estos archivos es el hecho de que son más compactos y eficientes que los nativos de Java, lo cual es un aspecto muy importante a tener en cuenta, pues el objetivo final son dispositivos móviles con recursos limitados.

Por otro lado, las bibliotecas core de Java que nos encontramos en Android son algo diferentes de las ofrecidas en *Java SE* y *Java ME*, aunque siguen guardando multitud de similitudes.

Framework de aplicaciones

El framework de aplicaciones es la capa que define la estructura de las aplicaciones Android. Ésta aporta los bloques de construcción de alto nivel que van a permitir desarrollar las aplicaciones, dotándolas de la funcionalidad que precisen. La mayoría de los componentes de esta capa son librerías Java que acceden a los recursos de las capas anteriores a través de la máquina virtual *Dalvik*.

Aplicaciones

La capa más alta de la arquitectura es la que contiene todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no, las nativas (programadas en C o C++) y las administradas (programadas en Java), las que vienen preinstaladas en el dispositivo y aquellas que el usuario ha instalado.

El usuario final únicamente será consciente de esta capa, pues es la que le proporciona las utilidades que él necesita. Sin embargo, si se utiliza Android no solo como usuario sino también como desarrollador, ese necesario contemplar esta capa y todas las que la preceden para saber que está ocurriendo, y que es lo que hace que todo funcione.

3.1.2 Bloques de construcción

Android proporciona varios bloques de construcción que todo desarrollador debe conocer y tener en cuenta a la hora de llevar a cabo la implementación de cualquier proyecto [14]. Concretamente, los bloques de construcción hacen referencia a los objetos definidos en el SDK de Android. Los más importantes son:

- *Activities*: Una *Activity* (actividad) es un componente que se corresponde con una pantalla de la interfaz de una aplicación, la cual permite interactuar con el usuario. Una aplicación puede definir diferentes *Activities* para manejar las distintas fases del programa. Cada *Activity* será responsable de guardar su propio estado para que pueda ser restablecido posteriormente como parte del ciclo de vida de la aplicación, pues son independientes unas de otras. Asimismo una *Activity* puede usarse para obtener información global de la aplicación.

- *Intents*: Un *Intent* (intención) es un mecanismo para definir una acción específica en el sistema, a través de la descripción de la propia acción, de la información necesaria para llevarla a cabo y de quién debe realizarla. Por ejemplo, existe un *Intent* específico para "sacar una foto", "llamar a casa", o "escribir un correo electrónico". Además un *Intent* se utiliza muy frecuentemente como medio para intercambiar información entre *Activities*.

- *Services*: Un *Service* (servicio) se trata de una tarea que se ejecuta en segundo plano sin la iteración del usuario, por lo que no posee interfaz gráfica. Un ejemplo sería un reproductor que ejecuta música cuando otra aplicación esta en primer plano interactuando con el usuario.

- *Broadcast Receivers*: Los *Broadcast Receivers* (receptores de transmisiones) son componentes que tienen el propósito de recibir y reaccionar ante eventos globales, como pueden ser la recepción de una llamada o un mensaje.

- *Content Providers*: Un *Content Provider* (proveedor de contenido) es un conjunto de datos envueltos en una API personalizada, a los cuales se puede acceder y leer. Es el mejor método para compartir datos globales entre aplicaciones. Un ejemplo claro sería la lista de contactos.

3.1.3 Ciclo de vida de una aplicación Android

Las interfaces de usuario basadas en ventanas han alcanzado un gran éxito y aceptación en los ordenadores personales. Sin embargo, no son las más adecuadas en dispositivos móviles, pues éstos poseen una serie de características muy distintas. Cuentan con pequeñas pantallas y recursos limitados, por eso lo que prima ante todo es la sencillez a la hora de utilizar cada aplicación.

En Android, cuando el usuario ejecuta una aplicación, ésta se carga y se sitúa en primer plano, ocupando toda la pantalla a excepción de la barra de estado. Desde esa aplicación abierta se podrán lanzar otras aplicaciones o pasar a otras pantallas de la misma aplicación. Todos estos programas y pantallas serán guardados en una pila a través del gestor de actividades del sistema. De esta manera el usuario podrá navegar por la pila y acceder a las pantallas abiertas con anterioridad a través del botón "Volver" del dispositivo. En general puede decirse que es semejante al funcionamiento de un navegador Web.

Como se indicó anteriormente, en Android cada una de las pantallas que forman la interfaz de usuario están representadas por una *Activity*, de ahí que se haga referencia a ellas como actividades más que como pantallas. De este modo, una aplicación Android será un conjunto de actividades más un proceso Linux que las contiene y gestiona. Además, existe la posibilidad de compartir actividades entre

diferentes aplicaciones, pues si ya se tiene una actividad que realiza una determinada función que puede ser útil en otra aplicación, no es necesario volver a implementarla de nuevo.

También cabe destacar que una aplicación puede estar viva a pesar de que el proceso Linux que la contiene haya sido destruido. El ciclo de vida de una actividad no está ligado al ciclo de vida de un proceso: los procesos son meros contenedores desechables de actividades.

Asimismo, una actividad Android puede encontrarse en diferentes estados [14]. El desarrollador no tiene control sobre el estado actual de su aplicación, pues es controlado por el sistema, pero sí que recibirá notificaciones cuando éste vaya a cambiar gracias a una serie de métodos. Esto permite llevar a cabo unas u otras acciones dependiendo de cómo se encuentre la aplicación. Ese conjunto de métodos es el siguiente:

- *onCreate()*: Este método es llamado cuando se inicia la actividad por primera vez y suele emplearse para realizar las inicializaciones oportunas en la interfaz de usuario.
- *onStart()*: A través de este método se indica que la actividad va a ser mostrada al usuario.
- *onResume()*: Se llama cuando la actividad puede comenzar a interactuar con el usuario. Suele ser un buen lugar para arrancar animaciones o música.
- *onPause()*: Se ejecuta cuando la actividad se dispone a pasar a segundo plano, normalmente debido a que otra actividad ha sido ejecutada en primer plano. Es en este punto donde se debe guardar el estado persistente de la aplicación.
- *onStop()*: Es llamado cuando la actividad ya no es visible para el usuario y no será necesaria durante un tiempo.
- *onRestart()*: Cuando este método es llamado se está volviendo a mostrar la actividad al usuario tras haber sido detenida.
- *onDestroy()*: Este método actúa justo en el instante antes de que la actividad sea destruida.

A continuación se muestra una ilustración en la que se puede observar todo el ciclo de vida de una actividad (Figura. 3.2).

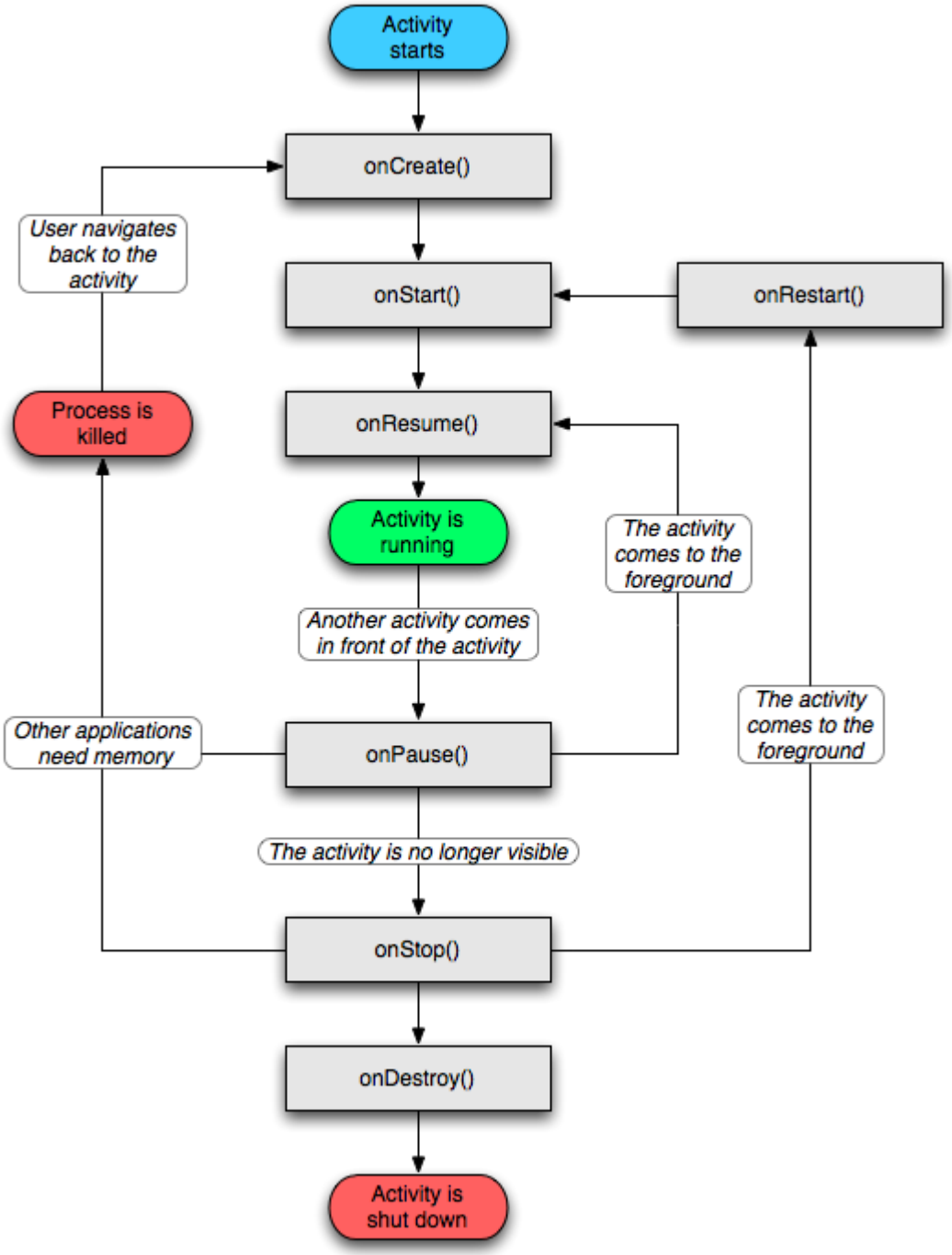


Figura 3.2 Ciclo de vida de una actividad Android.

3.2 Tecnologías para la comunicación entre dispositivos

Si se dejan a un lado los detalles técnicos que envuelven al sistema, se puede ver que, en líneas generales, lo que se desea implementar es una comunicación entre dos entes que intercambian una determinada información. Asimismo, en cualquier comunicación entre un emisor y un receptor es preciso disponer de un canal capaz de transmitir el mensaje desde el origen hasta el destino. Por lo tanto, la elección de una tecnología capaz de proporcionar un canal de comunicación entre dos dispositivos es una tarea ineludible.

De entre las diferentes alternativas existentes, se ha focalizado el interés en cuatro de ellas por las características que presentan. A continuación se describen brevemente.

Wi-Fi en modo infraestructura

Una red inalámbrica en modo infraestructura es una red tipo cliente-servidor, donde los dispositivos cliente se conectan a un *punto de acceso* (PA) que actúa como servidor. Para que pueda existir comunicación entre dos dispositivos cliente, ambos deben estar conectados al mismo punto de acceso. Asimismo, cuando un cliente quiere enviar un mensaje a otro dispositivo, lo envía al punto de acceso y este lo reenvía hasta el cliente destino, por lo que no existe un enlace punto-a-punto directo.

Como se puede ver, se trata de un sistema completamente centralizado, lo que dificulta notablemente su utilización en un entorno vehicular, bajo las necesidades y requisitos que precisa el sistema que se quiere implementar. Sin embargo, debido a que la puesta en marcha de este tipo de comunicación en dispositivos móviles es bastante simple y prácticamente inmediata, puede emplearse como solución a la hora de realizar pruebas y tests en el laboratorio.

AllJoyn Framework

AllJoyn es un proyecto de código abierto desarrollado por *Qualcomm Innovation Center, Inc.*, y patrocinado por la *AllSeen Alliance* [15]. Su propósito es proporcionar un marco de trabajo universal que permita la interoperabilidad entre productos tecnológicos de diversa índole: ordenadores, teléfonos, televisores, coches, etc., para ser capaces de construir redes dinámicas basadas en la proximidad entre nodos, sin necesidad de intermediarios y con independencia de cuál sea el fabricante de los dispositivos. Además, no precisa de un control exhaustivo sobre la tecnología de comunicación subyacente empleada: Wi-Fi, Bluetooth, 3G/4G, etc. Este proyecto intenta fomentar el desarrollo de aplicaciones en el ámbito de lo que se conoce como *Internet of Everything*, o Internet de las cosas.

El modus operandi de *AllJoyn* sigue el paradigma de *Invocación de Métodos Remotos* adoptado por otras tecnologías, donde los nodos de la red ofertan una serie de servicios que otros integrantes pueden utilizar a través de la llamada a un método de un objeto remoto, el cual se encuentra en el nodo que oferta dicho servicio.

A primera vista puede parecer una tecnología ideal para el problema planteado, pues bajo las premisas citadas en los párrafos anteriores permitiría establecer un canal de comunicación directo entre dos dispositivos de manera sencilla y rápida. Sin embargo, si se profundiza un poco en su funcionamiento, pueden vislumbrarse ciertos aspectos importantes que desaconsejan su utilización en este proyecto, tal y como se describe a continuación:

- Para determinar si varios dispositivos se encuentran cerca los unos de los otros, se apoya en los PA comunes que estos pueden ver. Dicho de otro modo, si varios dispositivos son capaces de descubrir un mismo PA, *AllJoyn* supone que éstos se encuentran próximos entre sí, y es capaz de establecer un canal de comunicación entre ellos. Por lo tanto, al apoyarse en esta idea, parece estar pensado para operar correctamente en redes de área local (LANs) y no tanto en redes vehiculares ad-hoc (VANETs), pues no existen PA en los que fijarse.

- Con la intención de solventar estas limitaciones, *AllJoyn* contempla una vía alternativa. Esta consiste en hacer uso de la tecnología de red con que cuentan los dispositivos de telefonía móvil, la tecnología 3G/4G. En este caso, el problema se encuentra en que puede ser necesario el uso de servidores intermedios para determinar la proximidad entre los dispositivos, o incluso para establecer el canal de comunicación. Por lo tanto, al no tratarse de una comunicación directa, podría suceder que el servicio prestado no fuera tan eficiente como cabría esperar.

- En relación con la idea anterior, *AllJoyn* proporciona soporte para el uso de tecnologías de comunicación directa, como son Bluetooth o Wi-Fi Direct, lo cual podría solventar el problema de los servidores intermedios. Sin embargo, Bluetooth no permite rangos de acción demasiado extensos, y la utilización de Wi-Fi Direct a través de *AllJoyn* no parece encontrarse aún en un estado lo suficientemente maduro.

- Asimismo, puede darse el caso de que el operador de telefonía que gestiona la red 3G/4G haya decidido bloquear ciertos paquetes de datos, como por ejemplo los UDP. En este escenario sería imposible establecer conexión entre los dispositivos, pues *AllJoyn* utiliza este tipo de paquetes en ciertas fases del proceso de conexión.

Wi-Fi Direct

Wi-Fi Direct o Wi-Fi P2P [16], es un estándar Wi-Fi que permite conectar entre sí aquellos dispositivos que cuenten con tecnología inalámbrica Wi-Fi, de una manera fácil y rápida. No precisa de la existencia de puntos de acceso, y permite establecer conexiones a velocidades idénticas a las de una comunicación Wi-Fi tradicional. La idea clave se encuentra en que Wi-Fi Direct embebe un punto de acceso software (Soft AP) en cada dispositivo preparado para dar soporte a este estándar.

Esta tecnología es capaz de conectar dispositivos de fabricantes diferentes, y lo único realmente necesario para establecer una comunicación *peer-to-peer* es que al menos uno de los dispositivos involucrados tenga soporte para Wi-Fi Direct. De este modo, el resto de dispositivos que integran la red (clientes del grupo) transmitirán la información de unos a otros a través del punto de acceso software (propietario del grupo), utilizando para ello el modo infraestructura de la tecnología Wi-Fi convencional.

Por sus características parece que esta tecnología encaja perfectamente en el ámbito del problema a resolver. Además, en este caso se lograría establecer un canal directo entre los dispositivos y se contaría con una API desarrollada directamente por Android. Sin embargo, al igual que ocurría en el caso de *AllJoyn*, existen ciertos matices que motivan la búsqueda de una tecnología alternativa:

- Wi-Fi Direct tiene soporte solo en las versiones más actuales de Android, exactamente a partir de Android 4.0 (API 14), por lo que si se intentaran conectar dos dispositivos con una versión de Android anterior sería imposible lograrlo, pues no existiría PA software alguno.

- Wi-Fi Direct solo permite establecer redes con topología estrella, es decir, con un nodo actuando como PA y el resto conectados a él como clientes. Además, un nodo solo puede pertenecer a una única red Wi-Fi Direct, comportándose como cliente o como PA. Esto implica que no está garantizado que todos los dispositivos puedan enviar y recibir vídeo al mismo tiempo en un entorno vehicular dinámico y multi-salto, lo cual limitaría bastante la funcionalidad de la aplicación.

Wi-Fi en modo ad-hoc

La tecnología Wi-Fi tradicional también proporciona un modo de funcionamiento descentralizado, el modo ad-hoc, en el cual no se precisa de la existencia de un PA que regule y gestione las comunicaciones dentro de la red: los nodos se comunican directamente unos con otros estableciendo enlaces punto-a-punto entre sí.

En la redes ad-hoc todos los nodos juegan el mismo papel, tratándose de una comunicación entre iguales (*peer-to-peer*) donde cada nodo es libre de asociarse con cualquier otro dispositivo, siempre que se encuentre dentro del rango de transmisión. Este tipo de red permite la adhesión de nuevos nodos por el solo hecho de que se encuentren dentro del rango de alcance de algún miembro ya perteneciente a la red.

En el caso particular que nos ocupa no tiene interés formar una red ad-hoc como tal, pues la información que se va a intercambiar no debe ser transmitida entre dos nodos que no estén directamente conectados a una distancia mayor que un salto. De este modo no se precisará de ningún algoritmo de encaminamiento, pues la información se transmitirá directamente a través del enlace punto-a-punto construido entre cada par de dispositivos que se encuentren próximos entre sí.

Además es capaz de solventar los problemas que presentaban las otras tecnologías:

- Permite establecer un canal de comunicación directo entres los entes de la comunicación, sin intermediarios.

- Un mismo dispositivo puede comunicarse con tantos otros dispositivos como desee, lo que quiere decir que no existe limitación en el numero de enlaces punto-a-punto que éste puede crear. Por lo tanto, se podrá enviar y recibir vídeo al mismo tiempo sin problemas.

- El modo ad-hoc de Wi-Fi está disponible en cualquier versión de Android, el único requisito es disponer de privilegios de administrador (*root*). Sin embargo, esos privilegios no vienen concedidos por defecto, por lo que es necesario que el usuario del dispositivo lleve a cabo una serie de modificaciones sobre el sistema operativo para obtenerlos. Esas modificaciones difieren en función del terminal y, en muchas ocasiones, pueden suponer la pérdida de la garantía otorgada por el fabricante .

Por consiguiente, tras lo expuesto en los párrafos anteriores, la solución más optima y que más se ajusta a las necesidades del sistema a implementar es la tecnología Wi-Fi en modo ad-hoc. La principal limitación de esta solución es que los dispositivos móviles deben poseer privilegios de administrador, lo cual reduciría la aceptación de la aplicación por parte de los usuarios, pues la gran mayoría son reacios a realizar cambios sustanciales en su dispositivo, o directamente carecen de los conocimientos para llevarlos a cabo.

No obstante, las tecnologías Wi-Fi Direct y Wi-Fi en modo infraestructura también pueden ser de utilidad en un entorno de pruebas, debido a que su puesta en marcha es mucho más simple y rápida.

3.3 Geolocalización

El término geolocalización hace referencia a la obtención de la ubicación geográfica que un objeto espacial ocupa dentro de un sistema de coordenadas determinado, con el fin de establecer un relación entre dicha localización y otro tipo de información, la cual permita proporcionar servicios más precisos, inteligentes, y con mejores prestaciones. Ya no se trata únicamente de información geográfica destinada a ser utilizada por los expertos de las ciencias geográficas y los *Sistemas de Información Geográfica* (SIG). Ahora la geolocalización tiene un impacto sociológico puesto que se aplica a todos los contenidos sociales presentes en el mundo.

En concreto, la geolocalización se ha convertido en una característica prácticamente imprescindible dentro de los dispositivos móviles inteligentes, pues ha hecho posible que se desarrollen aplicaciones que dan respuesta a problemas que no habían sido cubiertos hasta el momento, y ha permitido que se mejoren las funciones prestadas por otras aplicaciones ya existentes.

A grandes rasgos, existen dos tipos de aplicaciones principales dentro del ámbito de la geolocalización:

- Por un lado están aquellas que permiten adjuntar una ubicación al contenido creado por el usuario, como puede ser la acción de compartir el estado actual del tráfico en la vía en la que se encuentra un determinado conductor, o proporcionar información útil sobre el museo que se acaba de visitar.

- Por otra parte tenemos las aplicaciones que ayudan al usuario a decidir hacia dónde dirigirse, como por ejemplo aquellas que asisten al conductor de un vehículo guiándolo en la ruta hasta su destino, o aquellas que proporcionan una lista con los principales sitios de interés en función de la ubicación del usuario.

En el caso concreto de Android [17], cuando se lleva a cabo el desarrollo de una aplicación con reconocimiento de ubicación, el primer paso es elegir el proveedor de localización que satisface las necesidades del sistema. Hay que tener en cuenta que ciertas aplicaciones requieren localizaciones muy precisas y actualizadas, mientras que otras son menos restrictivas y operan en escenarios donde la geolocalización solo se utiliza en momentos puntuales. Es posible emplear tanto el GPS como el proveedor de localización por red de Android. Comúnmente, el GPS proporciona más exactitud aunque sólo funciona al aire libre, consume más rápido la batería, y puede tardar en devolver la ubicación más de lo que se desea. Por otro lado, el proveedor de localización por red determina la localización del dispositivo mediante la triangulación de señales de antenas de telefonía y las señales Wi-Fi, de manera que funciona tanto en interiores como en exteriores, su tiempo de respuesta es menor, y utiliza menos

energía. Asimismo, existe la posibilidad de no tener que escoger entre un método u otro, pues si la aplicación lo requiere se pueden utilizar ambos al mismo tiempo.

Independientemente del método elegido, obtener la ubicación de un dispositivo móvil es una tarea complicada. Existen varias razones por las que la lectura de la ubicación puede contener errores y ser inexacta. Los retos para determinar la ubicación del dispositivo incluyen:

1. **Múltiples fuentes de localización**: Como se ha visto, tanto el GPS como la localización por red proporcionan la ubicación del dispositivo. Para determinar cuál es la más adecuada en cada caso concreto, bastará con medir que es más importante: precisión, velocidad, o eficiencia energética.

2. **Movimiento del usuario**: Por norma general el dispositivo se encontrará en constante movimiento, lo que obliga a recalcular su ubicación cada cierto tiempo. La frecuencia de actualización dependerá de los requisitos de cada aplicación concreta.

3. **Precisión variable**: Las estimaciones de localización de cada una de las fuentes utilizadas no son consistentes en su precisión. Esto quiere decir que una ubicación obtenida 10 segundos antes desde una fuente determinada puede ser mucho más exacta que la localización más reciente de otra fuente.

En el caso concreto del sistema en que se centra este trabajo, la obtención de la ubicación del dispositivo es un aspecto crítico del mismo. La localización se utiliza para determinar donde se encuentran los vehículos cercanos y así seleccionar aquel situado inmediatamente delante, y en el mismo carril. Por consiguiente, la precisión es el parámetro más importante a tener en cuenta, aunque la frecuencia de actualización también juega un papel significativo, pues los vehículos se desplazan velozmente y una ubicación precisa pero antigua carecería de utilidad. Consecuentemente, el GPS es el proveedor de localizaciones que más se ajusta a las necesidades expuestas.

3.4 RTSP: Real Time Streaming Protocol

Cuando se reproduce un flujo de datos multimedia almacenado dentro de un CD o un DVD, el usuario es capaz de controlar la reproducción pudiendo pausarla o reanudarla, saltar a un punto anterior o posterior de la misma, o hacer un avance rápido o un rebobinado. Para conseguir esta misma funcionalidad a la hora de reproducir contenido multimedia que no se encuentra alojado localmente en el dispositivo reproductor, sino en un servidor remoto, es necesario un protocolo que permita intercambiar la información de control de la reproducción entre el cliente y el servidor. El protocolo de

flujos en tiempo real (RTSP, Real-Time Streaming Protocol), definido en *RFC 2326*, es dicho protocolo. RTSP permite que un reproductor multimedia cliente controle la transmisión de un flujo multimedia ofrecido por un servidor multimedia a través de la red, es decir, actúa de forma similar a un mando a distancia [18]. Sin embargo, es importante destacar que en el sistema implementado únicamente se hace uso de RTSP para iniciar y detener la sesión de streaming de vídeo, pues carece de sentido utilizar el resto de funcionalidades.

A continuación se describen las propiedades principales de este protocolo:

- Es un protocolo no orientado a conexión, en lugar de esto el servidor mantiene una sesión asociada a un identificador.

- Se trata de un protocolo fuera de banda, de manera que para una misma sesión mantiene dos canales de comunicación, uno para los datos de control y otro para el flujo multimedia. En la mayoría de los casos se emplea TCP en el canal de control y UDP para los datos de audio y vídeo aunque también se puede usar TCP si fuera necesario.

- Es un protocolo extensible, pues se le pueden añadir nuevos métodos y parámetros con facilidad.

- No define esquemas de compresión para audio y vídeo.

- No establece como se encapsula el audio y el vídeo para su transmisión a través de una red; RTP (*RFC 3550*) o cualquier otro protocolo propietario pueden proporcionar el mecanismo de encapsulación para los flujos multimedia.

- Posee capacidad multi-servidor, de tal manera que si un flujo multimedia se encuentra repartido entre diferentes servidores, el cliente establece varías sesiones de control, concurrentes entre sí, con los diferentes servidores.

- No restringe como el reproductor multimedia almacena en buffer el audio/vídeo. El flujo multimedia puede reproducirse tan pronto como empieza a llegar al cliente, puede reproducirse tras unos pocos segundos de retardo o puede descargarse por completo antes de empezar la reproducción.

- Es capaz de controlar dispositivos de grabación y reproducción (p. ej. cámaras IP RTSP).

Asimismo, el protocolo RTSP es similar en sintaxis y operaciones al protocolo HTTP, de forma que los mecanismos de expansión agregados a HTTP pueden, en muchos casos, añadirse a RTSP. Sin embargo, RTSP difiere de HTTP en un número significativo de aspectos:

1. Introduce nuevos métodos y posee un identificador de protocolo diferente.
2. Un servidor RTSP necesita mantener el estado de la sesión.
3. Tanto el servidor como el cliente pueden lanzar peticiones.
4. Los datos son transportados por un protocolo diferente (p. ej. RTP).

3.4.1 Peticiones RTSP

Para poder llevar a cabo el control sobre un determinado flujo multimedia, RTSP define un conjunto de peticiones que en la mayoría de los casos son enviadas desde el cliente hacia el servidor. Estas directivas presentan muchas similitudes con respecto a las especificadas en el protocolo HTTP. A continuación se describen brevemente las más importantes (Tabla 3.1).

Directiva	Sentido	Descripción
OPTIONS	C → S	Devuelve los tipos de peticiones que acepta el servidor.
DESCRIBE	C → S	Este método permite obtener una descripción del recurso multimedia apuntado por una URL RTSP. El servidor responde a esta petición con una descripción del recurso solicitado, donde se incluye una lista de los flujos multimedia que serán necesarios para la reproducción, además de otros datos.
SETUP	C → S	Esta petición contiene la URL del flujo multimedia y una especificación de cómo será transportado. Típicamente incluye un puerto para recibir los datos RTP (audio/vídeo), y otro para los datos de control RTCP (meta-datos). El servidor responde confirmando los parámetros escogidos y rellena las partes restantes, como son los puertos que utilizados por él.
PLAY	C → S	Esta directiva provocará que el servidor comience a transmitir el flujo multimedia especificado, para lo cual utilizará los puertos preestablecidos con la orden SETUP.
PAUSE	C → S	Permite detener temporalmente uno o todos los flujos multimedia, de manera que puedan ser restablecidos mas adelante con una orden PLAY.
RECORD	C → S	Hace posible que un cliente solicite la grabación de un flujo multimedia. El cliente debe indicar al servidor a través de una URL válida, donde almacenar los datos resultantes.

ANNOUNCE	C → S / S → C	Si es emitido por un cliente, permite enviar la descripción de un objeto multimedia hasta un servidor determinado. Si el que lo envía es el servidor, actualiza la descripción de una sesión RTSP en tiempo real.
TEARDOWN	C → S	Permite finalizar la sesión RTSP. Detiene la entrega de datos para la URL indicada liberando los recursos asociados.
GET_PARAMETER	C → S	Devuelve el valor de uno o varios parámetros de un determinado flujo multimedia.
SET_PARAMETER	C → S	Con esta directiva un cliente solicita establecer el valor de uno o varios parámetros de un flujo multimedia especifico.
OK	S → C	Lo emite el servidor como confirmación positiva a una petición previa realizada por un cliente.
REDIRECT	S → C	Permite que un servidor informe a un cliente de que el flujo multimedia solicitado se encuentra en una localización diferente.

Tabla 3.1 Directivas RTSP.

3.4.2 Sesión RTSP

Para entender mejor el funcionamiento del protocolo RTSP, es importante comprender lo que sucede durante una sesión de este protocolo. Por ello a continuación se describe un ejemplo simple de utilización de RTSP, el cual queda ilustrado en la Figura 3.3.

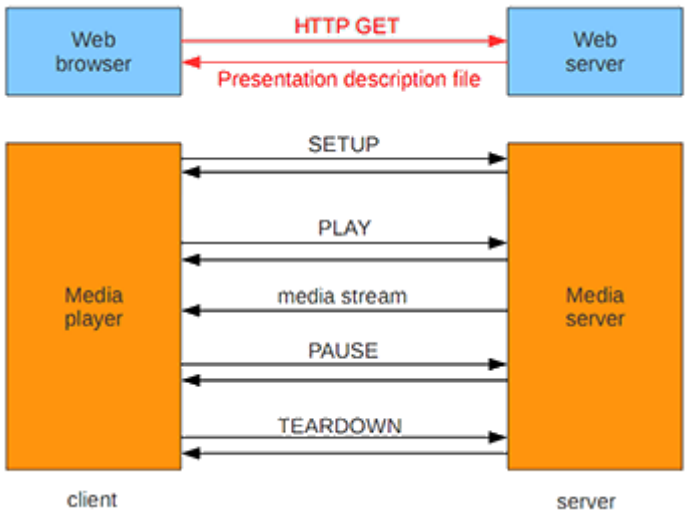


Figura 3.3 Iteración entre un cliente y un servidor en una sesión RTSP.

En primer lugar, el cliente a través de un navegador web solicita un archivo de descripción de la presentación a un determinado servidor web. Este archivo de descripción puede contener referencias a diversos archivos multimedia continuos, así como directivas para la sincronización de los mismos. Cada referencia a un archivo multimedia continuo se expresa a través de una URL del tipo *rtsp://*.

EL servidor web encapsula el archivo de descripción de la presentación en un mensaje de respuesta HTTP y lo envía al navegador web del cliente. Cuando el navegador recibe dicha respuesta HTTP, invoca a un reproductor multimedia basándose en el campo que define el tipo de contenido del mensaje. Seguidamente, el reproductor envía una solicitud RTSP *SETUP* al servidor multimedia donde se aloja el flujo de datos a reproducir. El servidor responde con un mensaje RTSP *OK* para confirmar que todo es correcto. Consecuentemente, el reproductor envía una solicitud RTSP de reproducción *PLAY*, y el servidor le responde con un mensaje RTSP *OK*. En este punto, el servidor comienza a retransmitir el flujo multimedia hacia el reproductor, al mismo tiempo que mantiene un control sobre la sesión en un canal paralelo. A continuación, el reproductor multimedia emite una solicitud RTSP de pausa con el mensaje *PAUSE* y el servidor responde de nuevo con un mensaje RTSP *OK*, lo que detiene temporalmente el flujo de datos. Cuando el usuario ha terminado, el reproductor multimedia envía una solicitud RTSP *TEARDOWN* para finalizar la sesión y el servidor confirma esta petición con un nuevo mensaje RTSP *OK*.

Para finalizar, se muestran los mensajes RTSP reales intercambiados entre el reproductor (C) y el servidor multimedia (S) del ejemplo anterior:

```
C:  SETUP rtsp://audio.example.com/twister/audio RTSP/1.0
    Cseq: 1
    Transport: rtp/udp; compression; port=3056; mode=PLAY
S:  RTSP/1.0 200 OK
    Cseq: 1
    Session: 4231
C:  PLAY rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0
    Range: npt=0-
    Cseq: 2
    Session: 4231
S:  RTSP/1.0 200 OK
    Cseq: 2
    Session: 4231
C:  PAUSE rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0
    Range: npt=37
    Cseq: 3
    Session: 4231
S:  RTSP/1.0 200 OK
    Cseq: 3
    Session: 4231
C:  TEARDOWN rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0
    Cseq: 4
    Session: 4231
S:  RTSP/1.0 200 OK
```

Figura 3.4 Ejemplo de sesión RTSP.

3.5 RTP: Real-time Transport Protocol

Además de un protocolo que permita controlar la reproducción del flujo multimedia (RTSP), es necesario otro protocolo más cercano al nivel de transporte, que se encargue del encapsulado de los datos multimedia para una correcta transmisión y recepción. Dicho protocolo es RTP [18], el cual trabaja en el nivel de sesión.

Al igual que RTSP, RTP se apoya en el concepto de sesión, de tal manera que mantiene una asociación entre las aplicaciones que se comunican empleando este protocolo. Una sesión es identificada a través de una dirección IP multicast y dos puertos: uno para el flujo multimedia y otro para los datos de control. Asimismo, cada flujo multimedia distinto es transmitido usando una sesión diferente. Por ejemplo, si se quisiera transmitir audio y vídeo serían necesarias dos sesiones separadas. Esto permite que los participantes de la sesión puedan reproducir únicamente audio o vídeo si lo desean.

RTP, al no ser un protocolo propietario, posibilita la interoperabilidad entre las distintas aplicaciones multimedia de red que lo utilicen, siendo empleado principalmente en sistemas de streaming multimedia junto a RTSP, y sistemas de videoconferencia en conjunción con H.323 o SIP (Session Initiation Protocol).

Habitualmente, RTP se ejecuta sobre UDP, pues reduce el tiempo de envío de los paquetes a través de la red, ya que en aplicaciones de audio y vídeo es más importante una transmisión rápida que la pérdida de unos pocos paquetes durante el recorrido. El emisor del flujo encapsulará un fragmento multimedia dentro de un paquete RTP, el cual a su vez irá contenido dentro un datagrama UDP y de un paquete IP. Así, el receptor del flujo multimedia extraerá el paquete RTP del datagrama UDP, para a continuación obtener el fragmento multimedia que será pasado a la aplicación reproductora. Esta última hará uso de las cabeceras del paquete RTP para decodificar y procesar los datos multimedia.

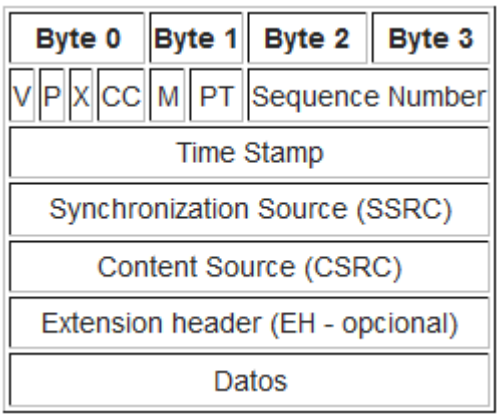


Figura 3.5 Paquete RTP.

Además, los paquetes RTP no están limitados a aplicaciones unidifusión, pudiendo también transmitirse siguiendo esquemas multidifusión *uno-a-muchos* y *muchos-a-muchos*. Dichos paquetes siguen la estructura mostrada en la Figura 3.5. A continuación se describe el contenido de cada uno de los campos que forman un paquete RTP:

- *Número de versión (V):* 2 bits. Versión de RTP.
- *Relleno (P):* 1 bit. Si el bit del relleno está activado, hay uno o más bytes al final del paquete que no son parte de la carga útil. El último byte del paquete indica el número de bytes de relleno. El relleno es usado por algunos algoritmos de cifrado.
- *Extensión (X):* 1 bit. Si el bit de extensión está activado, quiere decir que existe una extensión de la cabecera. Este mecanismo permite añadir información al encabezado RTP.
- *Conteo CSRC (CC):* 4 bits. Número de identificadores CSRC que contribuyen a los datos.
- *Marcador (M):* 1 bit. Definido por cada perfil multimedia particular.
- *Tipo de carga útil (PT):* 7 bits. Permite indicar el tipo de codificación de audio/vídeo utilizada.
- *Número de secuencia (Sequence Number):* 16 bits. Establece el orden de los paquetes de un mismo flujo multimedia. Permite que el receptor restaure la secuencia de paquetes y detecte posibles pérdidas.
- *Marca de tiempo (Time Stamp):* 32 bits. Indica el instante de captura del primer octeto del campo de datos.
- *SSRC:* 32 bits. Número asignado aleatoriamente por el emisor para identificar el origen del flujo RTP.
- *CSRC:* 32 bits. Identifica las fuentes contribuyentes para la carga útil. Puede haber más de 16 fuentes contribuyentes. Si hay fuentes contribuyentes múltiples, entonces la carga útil son los datos mezclados de esas fuentes.
- *Extensión de cabecera (EH):* CCx32 bits. Contiene información adicional que se suma al encabezado fijo.
- *Datos:* Carga útil.

Asimismo, RTP trabaja en colaboración con el protocolo de control de sesión RTCP (RFC 3550). RTCP es empleado para enviar datos de control entre el emisor y el receptor de una sesión RTP. Los paquetes RTCP no encapsulan datos multimedia, sino que contienen informes del emisor y/o receptor, que son emitidos periódicamente para anunciar estadísticas que pueden ser útiles para la aplicación emisora o receptora. Estas estadísticas incluyen el número de paquetes enviados, el número de paquetes perdidos y la fluctuación entre llegadas. La especificación de RTP no establece que debe hacer la aplicación con esta información de control, por lo que tal cometido recae sobre el desarrollador de la propia aplicación. Por ejemplo, los emisores pueden emplear los datos de control para modificar sus velocidades de transmisión, o también se puede emplear con propósitos de diagnóstico; por ejemplo, los receptores pueden determinar si existen problemas de carácter local, regional o global.

4. Análisis del sistema

Una vez que se conoce el ámbito de aplicación del proyecto, así como los principales elementos en los que se apoya, se puede decir que se han establecido las bases de éste. Por lo tanto es hora de pasar a la realización propiamente dicha.

El primer paso en ese proceso es describir con exactitud aquello que se quiere implementar, destacando las necesidades y requisitos impuestos por el sistema, y aplicando especial énfasis en las partes más críticas y de las cuales depende el éxito final del trabajo realizado.

4.1 Descripción del sistema

El sistema desarrollado pretende asistir al conductor de un vehículo en la realización de las maniobras de adelantamiento. A través de la reproducción de un vídeo en tiempo real que muestre la perspectiva de la vía desde el vehículo situado en el mismo carril e inmediatamente delante, es decir el vehículo que se desea adelantar. De esta manera, el conductor que va a efectuar la maniobra dispondrá de más información acerca de las condiciones de la vía, lo que le permitirá decidir si el adelantamiento es o no seguro, y por lo tanto efectuarlo con mayor probabilidad de éxito. En concreto, será especialmente útil en vías de doble sentido de circulación con un solo carril, y en circunstancias de escasa visibilidad.

Para lograr implementar dicha funcionalidad, el sistema se desarrolla a través de una aplicación para dispositivos móviles con tecnología Android, pues proporcionan las herramientas necesarias para capturar y transmitir vídeo en tiempo real, y cuentan con la capacidad de computo, de almacenamiento y de red necesarias para que el sistema se ejecute correctamente.

Asimismo, el sistema no debe suponer una distracción para el conductor en ningún caso, pues su objetivo final es mejorar la seguridad vial. Por consiguiente, la aplicación solo debe requerir de la intervención del usuario justo antes de arrancar el vehículo y tras estacionarlo, para activar y desactivar el sistema respectivamente. De igual modo, el dispositivo móvil deberá ir colocado en una posición donde el conductor sea capaz de visualizar perfectamente la pantalla sin apartar la vista de la carretera.

De este modo, una vez arrancado, el sistema operara en un segundo plano sin que el conductor del vehículo sea consciente de ello, comunicándose con los vehículos cercanos en busca de servir (servidor) y/o ser servido (cliente).

Figura 4.1 Ejemplo de colocación del dispositivo móvil dentro del vehículo.

Por lo tanto, existirán dos fases bien marcadas dentro del sistema. Una primera fase de comunicación o negociación, en la que los vehículos cercanos entre sí intercambian mensajes siguiendo un protocolo, con el fin de establecer comunicación entre los dos vehículos implicados en el adelantamiento: el que va a efectuar la maniobra y el que va a ser adelantado. Una vez establecido este emparejamiento, comenzará la segunda fase, en la que los dos vehículos, siguiendo un rol determinado (cliente o servidor), transmiten y reproducen el streaming de vídeo respectivamente (Figura. 4.2). Es importante destacar que el sistema es capaz de actuar como cliente y servidor de forma concurrente.

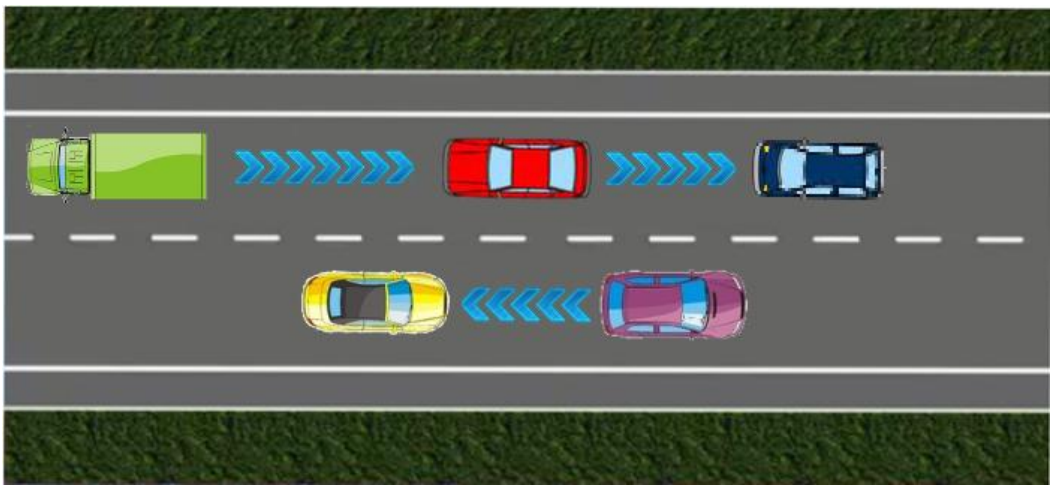


Figura 4.2 Transmisión del streaming de vídeo.

En resumen, en el sistema se diferencian 3 bloques funcionales principales:

1. Un protocolo de comunicación cliente-servidor que permite establecer la conexión entre los dos vehículos involucrados en el adelantamiento.

2. La captura, transmisión y reproducción de vídeo en tiempo real a través del protocolo RTSP.

3. La obtención de la ubicación del vehículo a través del sistema de geolocalización de Android para su uso en el protocolo C-S.

4.2 Requisitos del sistema

A continuación se presenta el conjunto de requisitos a cumplir por el sistema, a través de los cuales se pretende proporcionar una descripción más detallada de éste. Además, una vez finalizada la implementación, permiten comprobar si el sistema está funcionando correctamente y cumple con los objetivos marcados.

Existen muy diversas formas de categorizar los requisitos de un sistema, en el caso particular de este proyecto se ha decidido dividirlos en dos tipos principales siguiendo la clasificación de [19]:

1. **Requisitos Funcionales**: Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a cada tipo de entrada, y de cómo se debe comportar en situaciones concretas. En algunos casos, los requisitos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer.

2. **Requisitos No Funcionales**: Son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. Los requisitos no funcionales a menudo se aplican al sistema en su totalidad. Dentro de los requisitos no funcionales distinguimos tres subgrupos:

    - **Requisitos del producto**: Especifican el comportamiento del producto. Algunos ejemplos son los requisitos de rendimiento en la rapidez de ejecución del sistema y cuánta memoria se requiere; los requisitos de fiabilidad que fijan la tasa de fallos para que el sistema sea aceptable; los requisitos de portabilidad, y los requisitos de usabilidad.
    - **Requisitos organizacionales**: Estos requisitos se derivan de políticas y procedimientos existentes en la organización del cliente y en la del desarrollador. Algunos ejemplos son los estándares en los procesos que deben utilizarse; los requisitos de implementación, como los lenguajes de programación o el método de diseño a utilizar.
    - **Requisitos externos**: Incluyen todos los requisitos que se derivan de los factores externos del sistema y de su proceso de desarrollo. Éstos pueden incluir los requisitos de interoperabilidad que definen la manera en que el sistema interactúa con sistemas de otras organizaciones; los requisitos legislativos que deben seguirse para asegurar que el sistema funcione dentro de la ley, y los requisitos éticos.

4.2.1 Requisitos Funcionales

- La aplicación, una vez instalada, podrá ser lanzada a ejecución desde el menú de aplicaciones del dispositivo Android, a través de un icono perfectamente identificable.

- La interfaz gráfica de la aplicación ha de ser simple e intuitiva, de manera que permita una rápida configuración y puesta en marcha del sistema. Contendrá una barra de herramientas bien diferenciada que permitirá al usuario desplazarse entre las distintas actividades de la aplicación.

- El sistema solo requerirá de la atención del usuario para ser arrancado o detenido. Una vez en marcha operará en segundo plano sin molestar al usuario.

- Si el sistema se encuentra activado, deberá hacérselo saber al usuario a través de una notificación permanente situada en la barra superior del sistema.

- Mientras el sistema se encuentre activado, únicamente pasará a primer plano para reproducir el streaming de vídeo recibido desde un vehículo situado inmediatamente delante.

- El sistema implementará dos roles de funcionamiento completamente independientes entre sí, cliente y servidor. El servidor se encargará de capturar el streaming de vídeo y transmitirlo a un cliente situado inmediatamente detrás. El cliente deberá arrancar la reproducción del streaming recibido para que el conductor pueda visualizarlo.

- El sistema implementará un protocolo de comunicación cliente-servidor que permita a los dos vehículos involucrados en un adelantamiento establecer comunicación para transmitir/reproducir el streaming de vídeo.

- El sistema debe ser capaz de operar concurrentemente como cliente y como servidor, recibiendo vídeo de un coche situado inmediatamente delante y sirviendo a otro localizado inmediatamente detrás al mismo tiempo.

- El sistema debe ser capaz de obtener la ubicación del dispositivo móvil con una frecuencia y precisión suficientes como para permitir al protocolo C-S operar con las localizaciones reales de los vehículos.

- El sistema no debe molestar ni distraer al conductor del vehículo en ningún caso, por lo que no debe emitir sonidos, alertas, u otro tipo de acciones que puedan sobresaltarlo.

- El sistema debe ser capaz de adaptar la calidad del streaming de vídeo a la máxima resolución común de captura y reproducción entre el servidor y el cliente implicados en la comunicación, manteniendo siempre una calidad mínima que garantice una correcta visualización.

- El sistema transmitirá la velocidad del vehículo servidor junto con el streaming de vídeo de manera que el conductor del vehículo cliente pueda visualizarla al mismo tiempo que se reproduce el streaming. Esta información permitirá saber cuánto es preciso acelerar para sobrepasar al vehículo servidor.

- El streaming de vídeo debe contener únicamente imágenes, en ningún caso sonidos.

- La reproducción del vídeo se cerrará de forma automática una vez que el vehículo cliente haya adelantado al vehículo servidor.

- Una vez que el sistema haya sido detenido, deberá finalizar todos aquellos procesos que se ejecutan en segundo plano, dejando el sistema en un estado consistente.

4.2.2 Requisitos No Funcionales - Del Producto

- La aplicación deberá poder instalarse y ejecutarse en dispositivos móviles con una versión de Android igual o superior a la 4.0.x.

- La aplicación deberá poder instalarse en un dispositivo móvil compatible a través de un archivo en formato .apk.

- El tamaño final de la aplicación no excederá los 60 MB.

- El streaming de vídeo será codificado utilizando el codec H.264 y formato MP4. Asimismo, deberá reproducirse en el cliente con un retardo inferior a 1 segundo y a una tasa de entre 15 y 30 fps.

- Mientras el sistema se encuentre activado, el dispositivo permanecerá con la pantalla encendida y sin bloquear. Esto se debe a que existen dispositivos que deshabilitan ciertas funcionalidades de la tarjeta de red cuando se encuentran en standby, lo cual imposibilitaría el correcto funcionamiento del sistema.

- Para una correcta captura y reproducción del streaming de vídeo, el sistema requiere que el dispositivo móvil se encuentre colocado en posición horizontal y sobre un soporte ergonómico localizado en el salpicadero del vehículo.

4.2.3 Requisitos No Funcionales - Organizacionales

- La implementación del sistema se llevará a cabo a través del lenguaje de programación Java y bajo un sistema Linux que cuente con las siguientes herramientas:

  - IDE Eclipse para desarrollo en Java con el ADT de Android.
  - SDK de Android.
  - JRE y JDK (Versión más reciente).
  - Sistema de control de versiones Subversion (SVN)
  - Gimp (Para tratamiento de imágenes).

- Se emplearán los protocolo RTSP y RTP en la transmisión/reproducción del streaming de vídeo.

- Se empleará el protocolo de transporte UDP para la transmisión de los mensajes del protocolo C-S del sistema.

4.3 Protocolo de comunicación Cliente-Servidor

El protocolo de comunicación cliente-servidor (C-S) es sin duda el bloque principal del sistema, pues de él depende su correcto funcionamiento. El resto de elementos del sistema carecerían de sentido por si solos, pues, aunque proporcionan las funcionalidades esenciales, no podrían aplicarlas en tiempo y forma sin la ayuda del protocolo C-S.

El cometido del protocolo es ofrecer una comunicación ordenada y coherente entre dos dispositivos cercanos, cliente y servidor respectivamente. Permitirá emparejar a los dos vehículos involucrados en la maniobra de adelantamiento para que comiencen a transmitir y reproducir el streaming de vídeo según su situación.

Para comprender como está definido el protocolo y cuál es su funcionamiento es necesario conocer los diferentes estados por los que pasa el cliente y el servidor, los distintos mensajes intercambiados en cada uno de esos estados, así como las técnicas utilizadas para determinar la validez de un cliente o un servidor.

Como se mencionó en el apartado anterior, existen otros dos bloques principales que componen el sistema: el encargado de capturar, transmitir y reproducir el streaming de vídeo, y el sistema de geolocalización que permite obtener la ubicación de cada dispositivo. Sin embargo, no se mencionará

nada acerca de estos bloques en este capítulo, debido a que la información que se debe conocer de ellos es eminentemente técnica y su exposición se realizará en el *Capítulo 6* de implementación del sistema.

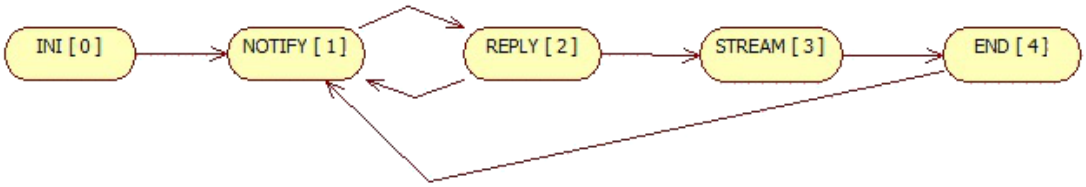
4.3.1 Diagramas de estados

Para comprender el funcionamiento del protocolo de comunicación implementado en el sistema es necesario conocer el comportamiento seguido por el cliente y el servidor dentro de él. Dichos comportamientos quedan definidos a través de dos diagramas de estados bien diferenciados.

Servidor

El servidor pasará por 5 estados diferentes:

[0] INI
[1] NOTIFY
[2] REPLY
[3] STREAM
[4] END



- **INI**: El servidor realiza la puesta a punto de los recursos que utilizará en el resto de estados. Pasará al estado *NOTIFY* en cuanto haya finalizado la inicialización de dichos recursos.

- **NOTIFY**: El servidor comienza a anunciar a los vehículos cercanos el servicio de streaming que ofrece. Para ello emitirá mensajes broadcast en los que indica la ubicación, dirección y sentido actuales del vehículo. La frecuencia de emisión de estos mensajes será de 1 segundo. A la vez que transmite los anuncios, permanecerá a la espera de alguna solicitud de conexión por parte de algún cliente. Pasará al estado *REPLY* al recibir una solicitud de conexión.

- **REPLY**: Sin dejar de emitir anuncios, el servidor evalúa la solicitud de conexión recibida. Dicha solicitud contendrá la ubicación, dirección, sentido y resolución máxima soportada del cliente. Si no es un cliente válido el servidor le enviará un mensaje de rechazo de conexión y volverá de nuevo al estado *NOTIFY*. En caso de ser un cliente válido, se establece la resolución del streaming de vídeo, intentando que ésta sea lo más cercana a la indicada por el cliente. Seguidamente el servidor enviará un mensaje de aceptación de conexión al cliente, en el que le indica el puerto de escucha del servidor RTSP, y pasará al estado *STREAM*.

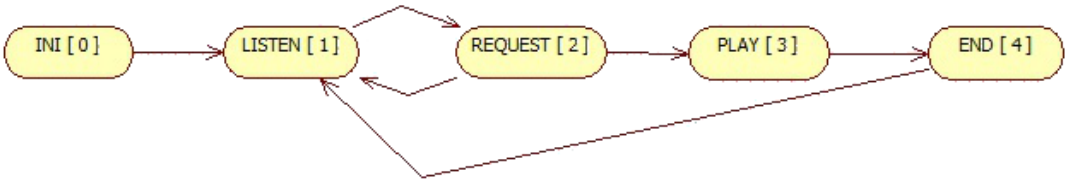
- **STREAM**: El servidor dejará de emitir anuncios y comenzará a transmitir el streaming de vídeo RTSP al cliente. Al mismo tiempo, a través de un canal paralelo, enviará mensajes al cliente con la velocidad, ubicación, dirección y sentido del vehículo. Cada uno de estos mensajes deben ser respondidos por el cliente para que el servidor considere que la sesión sigue activa. En caso de no recibir respuesta del cliente, el servidor no enviará más mensajes, cerrará la sesión, y pasará al estado *END*. Asimismo, el servidor puede recibir en cualquier momento un mensaje de fin de sesión por parte del cliente, y en ese caso se procede de igual modo que cuando no se recibe respuesta.

- **END**: El servidor liberará los recursos utilizados durante la sesión de streaming y retornará al estado *NOTIFY.*

Cliente

El cliente pasará por otros 5 estados:

[0] INI
[1] LISTEN
[2] REQUEST
[3] PLAY
[4] END



- **INI**: Al igual que el servidor, el cliente realiza la puesta a punto de los recursos que utilizará en el resto de estados. Pasará al estado *LISTEN* en cuanto haya finalizado la inicialización de dichos recursos.

- **LISTEN**: El cliente permanecerá a la espera de algún anuncio emitido por los servidores cercanos. Al recibir un anuncio el cliente analizará si se trata de un servidor válido; en caso negativo continuará esperando, y en caso positivo anotará al servidor como posible candidato y permanecerá a la escucha de más anuncios durante 2 segundos. Tras ese periodo de tiempo el cliente escogerá al candidato más optimo y pasará al estado *REQUEST*.

- **REQUEST**: El cliente envía una solicitud de conexión al servidor, la cual contendrá la resolución máxima soportada por el dispositivo móvil, así como la ubicación, dirección y sentido actuales del vehículo. A continuación se mantendrá a la espera de la contestación del servidor. Si el servidor acepta su petición el cliente pasará al estado *PLAY*, mientras que si la rechaza retornará al estado *LISTEN*.

- **PLAY**: El cliente se conectará al servidor RTSP y comenzará a recibir y reproducir el streaming de vídeo. Al mismo tiempo escuchará los mensajes de control emitidos por el servidor. La información contenida en estos mensajes le permitirá actualizar la velocidad del vehículo servidor en la pantalla de reproducción del streaming, y evaluar si se ha adelantado o no al servidor. En caso negativo se envía un reconocimiento al servidor para continuar con la sesión, en caso afirmativo el vídeo ya no es de utilidad, por lo que se envía un mensaje de fin de sesión al servidor y se pasa al estado *END*.

- **END**: El cliente liberará los recursos utilizados durante la sesión de streaming y retornará al estado *LISTEN.*

4.3.2 Tipos de mensajes

Además de los distintos estados en los que pueden encontrarse el cliente y el servidor, el otro elemento esencial del protocolo son los mensajes intercambiados entre ambos, los cuales contienen la información necesaria para que las dos partes validen si el otro es el cliente/servidor que buscan, así como para iniciar, mantener y finalizar la sesión de streaming de vídeo.

En concreto, el protocolo de comunicación cuenta con siete tipos de mensajes diferentes (Tabla 4.1).

Mensaje	Sentido	Estado Cliente	Estado Servidor	Contenido	Descripción
HELLO	S → C	LISTEN	NOTIFY	Ubicación, dirección, sentido.	Permite al servidor anunciar el servicio de streaming de vídeo que ofrece.
REQUEST	C → S	REQUEST	NOTIFY	Ubicación, dirección, sentido, resolución máxima soportada.	Lo emite el cliente para solicitar la conexión al servicio de streaming de vídeo.
READY	S → C	REQUEST	REPLY	Puerto de escucha servidor RTSP.	Respuesta positiva del servidor a una solicitud de conexión.
REJECT	S → C	REQUEST	REPLY	-	Respuesta negativa del servidor a una solicitud de conexión.
DATA	S → C	PLAY	STREAM	Ubicación, dirección, sentido, velocidad.	Permite indicar al cliente la información referente al vehículo servidor durante la sesión de streaming de vídeo.

DATA_ACK	C → S	PLAY	STREAM	-	Respuesta a un mensaje DATA. Hace posible que el servidor sepa que la conexión sigue viva.
END	C → S	PLAY	STREAM	-	Permite finalizar la sesión de streaming de vídeo.

Tabla 4.1 Tipos de mensajes del protocolo de comunicación C-S.

De este modo, el intercambio habitual para una sesión del protocolo de comunicación sería el representado en la Figura 4.3.

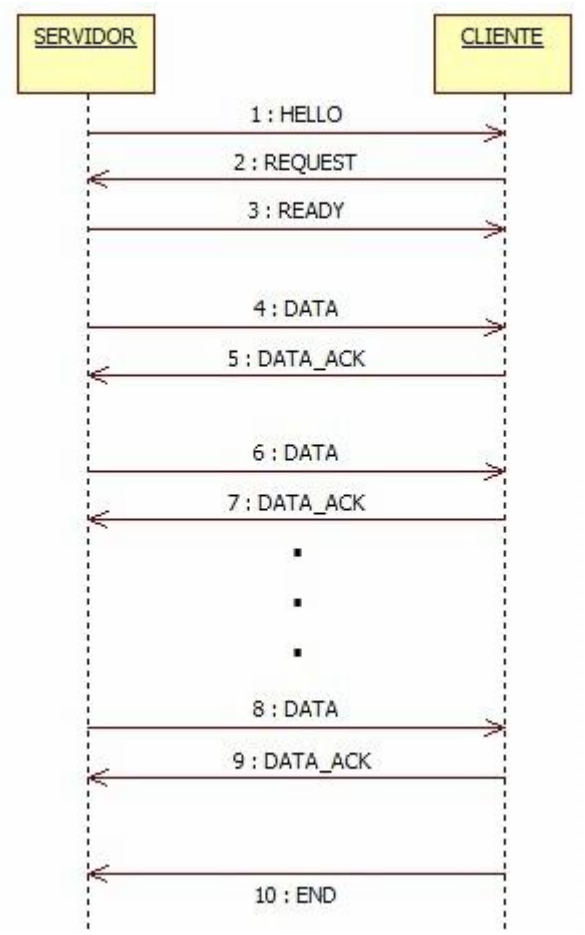


Figura 4.3 Ejemplo de sesión del protocolo de comunicación C-S.

4.3.3 Validación de localizaciones

El último bloque que completa el protocolo de comunicación es el mecanismo de validación utilizado por el cliente y el servidor para determinar si el otro sujeto de la comunicación está involucrado en la maniobra de adelantamiento. Así, el cliente validará que el servidor del que ha recibido un anuncio se encuentra situado inmediatamente delante de él y dentro del mismo carril, mientras que el servidor comprobará que el cliente del que ha recibido una solicitud de conexión se encuentra detrás y en el mismo carril.

Para lograrlo, cliente y servidor analizan la ubicación, dirección y sentido actuales del vehículo con el que están conversando. Como se vio anteriormente, esta información irá contenida en los mensajes del protocolo, concretamente a través de un vector de dos dimensiones $\overrightarrow{AB}$ (vector desplazamiento), siendo el punto B la ubicación más actual del vehículo (Figura 4.4).

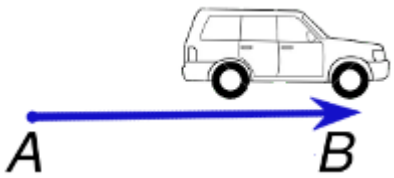


Figura 4.4 Vector desplazamiento del vehículo.

En primer lugar, tanto cliente como servidor comprobarán si el vehículo con el que están tratando sigue aproximadamente su misma dirección y sentido. Para ello analizarán el ángulo θ existente entre los vectores desplazamiento de ambos vehículos, y si dicho ángulo se encuentra por debajo de un determinado umbral α, se podrá concluir que efectivamente tienen la misma dirección y sentido (Figura 4.5).

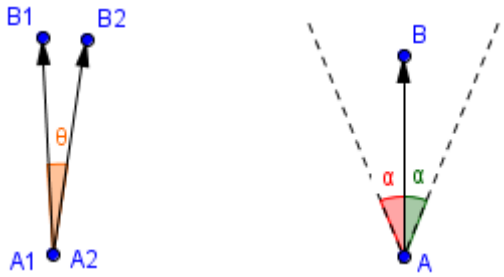


Figura 4.5 Validación de dirección y sentido.

A continuación, si la validación anterior fue satisfactoria, cliente y servidor comprueban si el otro vehículo se encuentra dentro del mismo carril de circulación y delante/detrás de su ubicación. Para tal fin, compararán el ángulo θ formado por su vector desplazamiento y el vector que une su ubicación actual con la del otro vehículo. Si el ángulo es menor que un umbral β, se deducirá que ciertamente se encuentra delante/detrás y en el mismo carril (Figura 4.6).

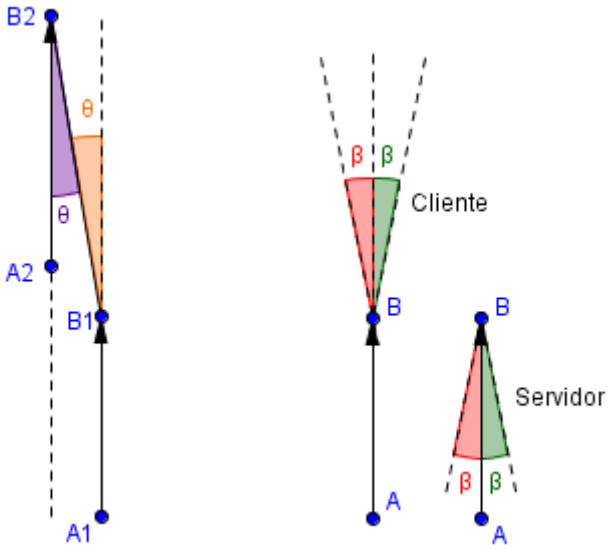


Figura 4.6 Validación de ubicación y carril.

Asimismo, el cliente deberá realizar una comprobación adicional, pues es posible que reciba anuncios de varios vehículos cercanos, situados delante y dentro del mismo carril, por lo que deberá seleccionar a aquel situado inmediatamente delante. Por ende, el cliente calculará la distancia desde su ubicación actual hasta los distintos candidatos, y seleccionará a aquel que se encuentre más cerca.

De esta manera, si se superan las validaciones descritas anteriormente, cliente y servidor pueden iniciar la transmisión/reproducción del streaming de vídeo, respectivamente.

Además, durante el transcurso de la sesión, el cliente evaluará si ya se ha sobrepasado al servidor, pues en ese caso el vídeo no es de utilidad y se puede finalizar la sesión. Para ello hará uso de la información contenida en los mensajes *DATA* emitidos por el servidor, calculando el ángulo θ formado por el vector desplazamiento del cliente y el vector que une la posición actual del vehículo cliente con la posición actual del vehículo servidor. Si el ángulo es mayor que un umbral φ, querrá decir que el vehículo cliente se encuentra por delante del vehículo servidor (Figura 4.7).

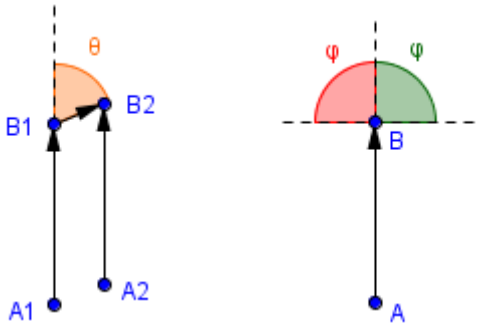


Figura 4.7 Validación de adelantamiento realizado.

Como se ha descrito en los párrafos anteriores, las distintas validaciones se apoyan esencialmente en el cálculo del ángulo formando por dos vectores. Por tanto, suponiendo dos vectores cualesquiera $\vec{u} = (u_1, u_2)$ y $\vec{v} = (v_1, v_2)$ con origen en el centro de coordenadas $(0,0)$, el ángulo α formado por ambos se obtiene aplicando la ecuación (4.1).

$$\alpha = \cos^{-1}\left(\frac{u \cdot v}{\|u\| \cdot \|v\|}\right) = \cos^{-1}\left(\frac{u_1 \cdot v_1 + u_2 \cdot v_2}{\sqrt{{u_1}^2 + {u_2}^2} \cdot \sqrt{{v_1}^2 + {v_2}^2}}\right) \qquad (\mathbf{4.1})$$

Al mismo tiempo, como los vectores desplazamiento de los vehículos no se encuentran centrados en el origen de coordenadas, será necesario desplazarlos hasta dicho punto antes de aplicar la ecuación (4.1). Así, para desplazar un vector cualquiera $\vec{u} = [(x_1, y_1), (x_2, y_2)]$ hasta un punto $P = (p_x, p_y)$, será preciso actualizar las coordenadas de $\vec{u}$ atendiendo a la expresión (4.2).

$$\left.\begin{array}{l} desplazamientoX = p_x - x_1 \\ desplazamientoY = p_y - y_1 \end{array}\right\} \begin{array}{l} x_1 = x_1 + desplazamientoX \\ y_1 = y_1 + desplazamientoY \\ x_2 = x_2 + desplazamientoX \\ y_2 = y_2 + desplazamientoY \end{array} \qquad (\mathbf{4.2})$$

También es preciso realizar el cálculo de la distancia entre dos coordenadas geográficas, pues como se ha indicado, el cliente debe ser capaz de seleccionar al vehículo servidor situado inmediatamente delante de él. Sin embargo no será necesario implementar dicho cálculo, pues el API de Android proporciona una función directa para obtenerlo.

5. Diseño del sistema

Después de determinar cuál ha de ser el comportamiento y las funcionalidades del sistema, es el momento de diseñar los componentes que lo hagan realidad.

En la fase de diseño se deben describir, de manera clara y precisa, todos los elementos que van a formar parte del sistema, para que a la hora de implementarlo queden cubiertos todos los requisitos y el funcionamiento sea el esperado.

Por lo tanto, en este capítulo se llevan a cabo tres tareas principales: definir la arquitectura del sistema, presentar las *clases* que componen dicha arquitectura a través de un *diagrama de clases*, y describir detalladamente el cometido y la estructura de cada una de ellas.

5.1 Arquitectura del sistema

La arquitectura del sistema permite describir a grandes rasgos, y con un alto nivel de abstracción, el conjunto de componentes que lo constituyen, así como las relaciones que mantienen entre ellos.

La arquitectura global del sistema no sigue un modelo prediseñado concreto. Sin embargo, sí toma algunos de los principios propuestos por el patrón de arquitectura de software *Modelo-Vista-Controlador* (MVC) [20]. En líneas generales, se ha dividido en una serie de bloques poco acoplados entre sí que desempeñan funciones bien diferenciadas dentro del sistema, intentado separar los datos y la lógica de negocio de la interfaz de usuario y del módulo encargado de gestionar los eventos y las comunicaciones de entrada/salida. En la figura 5.1 se muestra el diagrama de componentes que define la arquitectura del sistema.

A continuación se describe el cometido llevado a cabo por cada uno de esos componentes:

- **Vista-Controlador:** Este bloque de componentes tiene la importante tarea de implementar la comunicación entre el usuario y la lógica del sistema. Por una parte se encarga de generar una *vista* (interfaz) que el usuario pueda comprender y que muestre el estado del sistema. Por otro lado, se preocupa de atender las entradas del usuario (eventos sobre la vista), llevando a cabo las acciones oportunas sobre el *modelo*. Estos dos cometidos se encapsulan en una serie de *Activities*, cada una de las cuales posee un oyente específico o *controlador*, totalmente desacoplado de la generación de la interfaz de usuario. Asimismo, se hará uso de un componente adicional, el *Reproductor RTSP*, que

será utilizado por la *Activity* encargada de mostrar al usuario el streaming de vídeo emitido por un servidor.

- **Modelo:** Agrupa aquellos elementos que permiten confeccionar el comportamiento del sistema (*lógica*), como son: el protocolo de comunicación, los mecanismos de validación, la captura y transmisión del streaming de vídeo, y la obtención de ubicaciones. Para ello seguirá una arquitectura interna Cliente-Servidor, donde cada rol opera de forma independiente. Asimismo, para poder encapsular y transmitir adecuadamente el flujo de datos multimedia, se empleará un componente independiente, el *Servidor RTSP.*

- **Contexto:** El cometido de este componente es agrupar, en un mismo punto, elementos que se utilizan en varios de los bloques que forman el sistema. Así se evita tener instancias de *clases* repetidas, cuando con una para toda la aplicación es suficiente, y se ofrece un acceso centralizado y rápido a aquellos *métodos* comunes.

- **Base de Datos:** Proporciona la persistencia para los datos no volátiles del sistema. En concreto se utilizará para almacenar los *logs* (registros) cliente/servidor de la última sesión de streaming, de tal manera que puedan ser visualizados por el usuario cuando éste lo solicite.

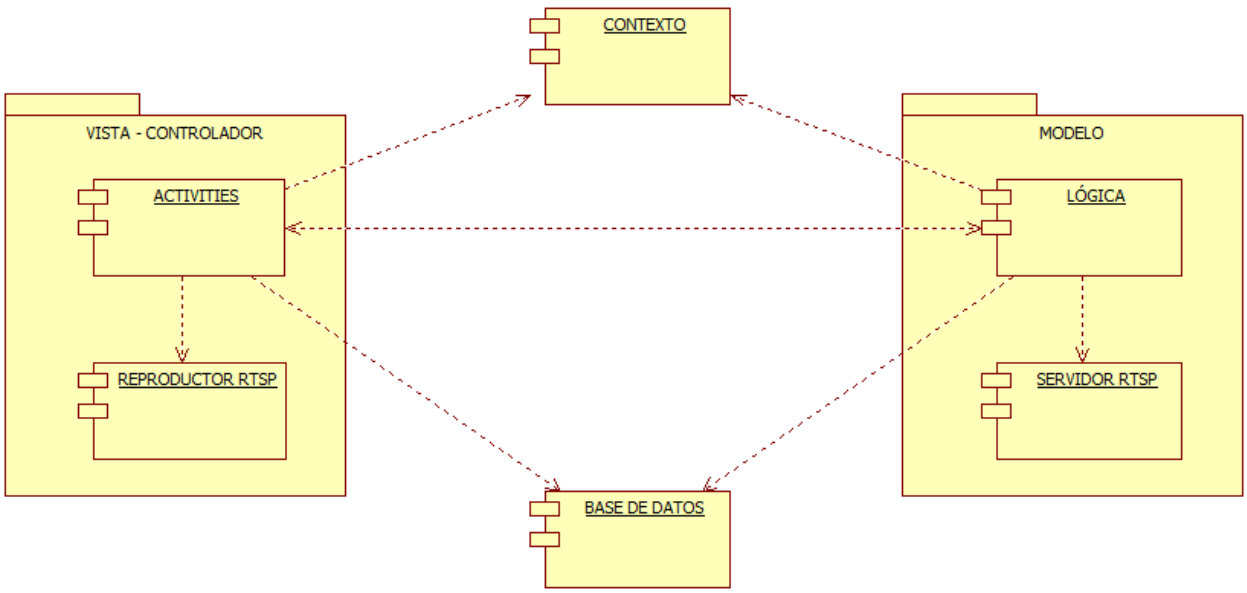


Figura 5.1 Diagrama de componentes del sistema.

5.2 Diagrama de clases

El diagrama de clases del sistema proporciona una visión global de las clases y objetos que lo componen, así como las relaciones que existen entre dichos elementos. Las clases deben ser consecuentes con los requisitos y el comportamiento fijados para el sistema en la fase de análisis, de tal manera que se garantice su correcto funcionamiento al implementarlo siguiendo este diseño.

Por lo tanto, en esta sección se va a mostrar el diagrama de clases (Figura 5.2) que, junto con la arquitectura descrita antes, definen el modelo de diseño del sistema. En el diagrama no se muestran los atributos y métodos de las clases, ni tampoco todas las clases internas, para que así pueda analizarse con mayor facilidad. En la siguiente sección se hablará en detalle de cada clase, y se abordarán también estos aspectos omitidos.

A modo de recordatorio, se presentan a continuación (Tabla 5.1) las distintas relaciones que pueden darse entre los elementos de modelado, definidas por el *Lenguaje Unificado de Modelado* (UML) [21], muchas de las cuales se utilizan en el diagrama de clases del sistema.

Relación	Símbolo	Descripción
Dependencia		Relación semántica en la cual un cambio a un elemento (independiente) puede afectar a la semántica del otro elemento (dependiente).
Asociación		Relación estructural entre clases que describe un conjunto de conexiones entre objetos que son instancias de las clases. Indica que los objetos que forma la asociación, están relacionados entre sí durante un periodo de tiempo continuado.
Agregación		Se trata de un caso concreto de asociación, que permite modelar objetos de mayor complejidad en base a relaciones todo-parte. En concreto, la agregación es una relación dinámica, donde el tiempo de vida del objeto que se agrega es independiente del objeto agregador.
Composición		Es similar a la agregación, con la diferencia de que en este caso se trata de una relación estática, donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del objeto compuesto.
Generalización		Relación de especialización/generalización en la que la clase especializada (hijo) extiende a la especificación de la clase generalizada (padre). Las subclases (hijos) comparten la estructura y el comportamiento de la superclase (padre).
Realización		Relación semántica entre elementos de modelado, donde uno de los elementos especifica un contrato que el otro elemento garantiza que cumplirá.

Tabla 5.1 Relaciones UML.

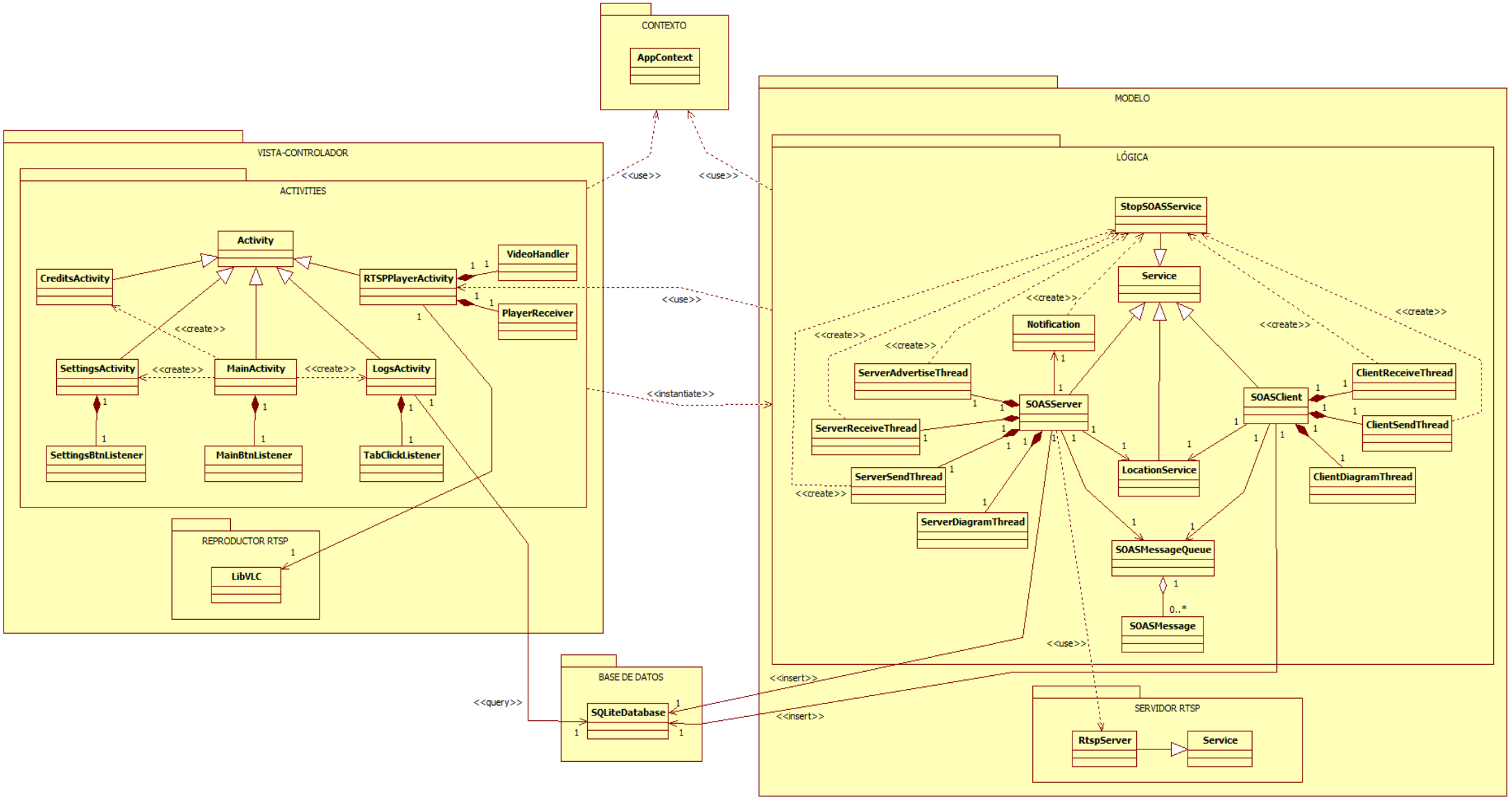


Figura 5.2 Diagrama de clases del sistema.

5.3 Descripción de clases

Una vez presentadas las diferentes clases que forman el sistema, así como las relaciones existentes entre ellas, es posible pasar a describirlas una por una. Esto permitiría comprender con más claridad cuál es el papel que juegan dentro del sistema, y así interpretar mejor el diagrama anterior.

Se profundizará más en unas clases que en otras, dependiendo de la relevancia de las acciones que llevan a cabo. Se describirán sus principales atributos y la función realizada por sus métodos.

Esta sección se estructurará siguiendo los bloques definidos en la arquitectura del sistema, para así tratar al mismo tiempo todas las clases con un cometido similar.

5.3.1 Vista-Controlador

- **MainActivity:** *Activity* lanzadora de la aplicación. Contiene un botón interruptor de dos posiciones que permite activar o desactivar el sistema de asistencia en adelantamientos (SOAS, *Safe Overtaking Android System*). Además, proporciona acceso a otras de las *Activities* que completan la vista: *SettingsActivity*, *LogsActivity*, *CreditsActivity*.

Atributos Destacados	
Atributo	**Descripción**
switchButton	Botón de dos posiciones que permite activar/desactivar el sistema.
settingsButton	Botón que da acceso a la pantalla de ajustes (SettingsActiviy).
logsButton	Botón que da acceso a la pantalla de visualización de logs (LogsActivity).

Métodos Destacados	
Método	**Descripción**
onCreateOptionsMenu(Menu)	Inicializa el contenido del menú de la Activity, el cual da acceso a la pantalla de créditos (CreditsActivity).
onOptionsItemSelected(MenuItem)	Método que ejecuta el oyente del menú cuando alguno de los elementos que contiene es pulsado. Debe arrancar CreditsActivity.

Clases Internas	
Clase	**Descripción**
MainBtnListener	Oyente encargado de gestionar los eventos que se producen sobre los botones de la Activity.
SwitchListener	Oyente encargado de gestionar los cambios de estado del botón interruptor que activa/desactiva el sistema.
StopSOASReceiver	Oyente que es notificado de la detención del sistema. En ese caso debe actualizar la interfaz de la Activity.

- **SettingsActivity:** *Activity* que permite configurar los parámetros de validación empleados por el sistema (SOAS). Estos son: el umbral para validar la dirección y el sentido de la marcha, el umbral para validar la ubicación, y el umbral para validar si se realizó el adelantamiento. Todos ellos son expresados en grados sexagesimales.

Atributos Destacados	
Atributo	**Descripción**
dirDegrees	Campo de texto numérico donde insertar el umbral de validación de la dirección y sentido de la marcha.
locDegrees	Campo de texto numérico donde insertar el umbral de validación de la ubicación.
overDegrees	Campo de texto numérico donde insertar el umbral de validación de adelantamiento realizado.
saveBtn	Botón que permite guardar los parámetros introducidos.
restoreBtn	Botón que permite restablecer los parámetros por defecto.
disableBtn	Botón de dos posiciones que permite deshabilitar la validación del sistema.

Métodos Destacados	
Método	**Descripción**
updateSettings()	Permite restablecer los ajustes guardados por el usuario.

Clases Internas	
Clase	**Descripción**
SettingsBtnListener	Oyente encargado de gestionar los eventos sobre los botones de la Activity.
CheckBoxListener	Oyente encargado de gestionar los cambios de estado del botón que habilita/deshabilita el sistema.

- **LogsActivity:** *Activity* que muestra los registros (*logs*) del cliente y del servidor, los cuales contienen los eventos ocurridos durante la última sesión SOAS.

Atributos Destacados	
Atributo	**Descripción**
tabC	Pestaña que permite cargar el log del cliente.
tabS	Pestaña que permite cargar el log del servidor.
dataBase	Objeto que permite hacer consultas sobre la base de datos, con el fin de obtener el log seleccionado por el usuario.

Métodos Destacados	
Método	**Descripción**
onCreateOptionsMenu(Menu)	Inicializa el contenido del menú de la Activity, el cual permite eliminar los logs del cliente y del servidor.
onOptionsItemSelected(MenuItem)	Método que ejecuta el oyente del menú cuando alguno de los elementos que contiene es pulsado. Debe eliminar de la base de datos los logs de eventos.
showLogs(int)	Permite cargar el log indicado como argumento. 1- Log cliente, 2-Log servidor.

Clases Internas	
Clase	**Descripción**
TabClickListener	Oyente encargado de gestionar los eventos que se producen sobre las pestañas de la Activity.

- **CreditsActivity:** *Activity* que muestra información relativa a la aplicación. Desarrollador, universidad, etc.

- **RTSPPlayerActivity:** *Activity* encargada de reproducir el streaming de vídeo RTSP emitido por un Servidor, para lo cual se apoya en la librería *libVLC* [22] [23].

Atributos Destacados	
Atributo	**Descripción**
streamingPath	Contiene la ruta al servidor RTSP que sirve el streaming de vídeo.
Libvlc	Reproductor multimedia.
videoWidth	Ancho del vídeo en pixeles.
videoHeight	Alto del vídeo en pixeles.

Métodos Destacados	
Método	**Descripción**
setSize(int, int)	Permite ajustar la resolución del vídeo (indicada como argumento) en función de las dimensiones de la pantalla del dispositivo.
createPlayer(String)	Crea un nuevo reproductor multimedia. El cual se conecta al servidor RTSP indicado como argumento, y arranca la reproducción.
releasePlayer()	Permite eliminar el reproductor multimedia una vez que no es de utilidad.

Clases Internas	
Clase	**Descripción**
VideoHandler	Oyente encargado de gestionar los eventos que se producen al reproducir el streaming de vídeo.
PlayerReceiver	Oyente que recibe dos tipos de notificaciones desde el cliente del sistema: 1- Ante la finalización de la sesión de streaming. En ese caso debe cerrar el reproductor multimedia. 2- Ante una actualización de la velocidad del servidor. En este caso debe mostrar en la pantalla del dispositivo la nueva velocidad.

5.3.2 Modelo

- **SOASServer:** *Service* que implementa el rol Servidor del sistema. Para poder capturar y transmitir el streaming de vídeo hace uso de la librería *libStreaming* [24], concretamente a través de la clase *RtspServer*.

Atributos Destacados	
Atributo	**Descripción**
state	Estado actual del servidor.
messageQueue	Cola de mensajes recibidos.
lService	Objeto que posibilita la comunicación con el servicio de ubicaciones (LocationService).
lastLocation	Vector desplazamiento más actual del vehículo.
rtspPort	Puerto de escucha del servidor RTSP.
waveLock	Objeto que permite mantener la CPU activa mientras el servidor este en marcha.

Métodos Destacados	
Método	**Descripción**
launchThreads()	Lanza a ejecución los distintos hilos secundarios de los que consta el servidor.
stopThreads()	Interrumpe los hilos secundarios.
getAvailablePort()	Devuelve un puerto que no se encuentre actualmente en uso, para arrancar en él el servidor RTSP.
putNotification()	Crea una notificación que indica al usuario que el sistema esta encendido, y que permite detenerlo cuando es pulsada.
startRTSPServer(int, int[])	Permite arrancar el servidor RTSP en un puerto determinado y con una resolución de vídeo concreta.
stopRTSPServer()	Permite detener el servidor RTSP.
getLocation()	Devuelve el vector desplazamiento más actual del vehículo.
getSpeed()	Devuelve la velocidad actual del vehículo.

Clases Internas	
Clase	**Descripción**
LocationServiceConn	Monitoriza el estado del servicio de ubicaciones (LocationService).
ServerHandler	Oyente encargado de recibir, evaluar y gestionar adecuadamente, los mensajes emitidos desde los hilos secundarios hacia el hilo principal del servidor.
ServerAdvertiseThread	Hilo emisor de notificaciones, encargado de anunciar a los dispositivos cliente el servicio de streaming RTSP que ofrece el servidor.
ServerSendThread	Hilo emisor de mensajes.
ServerReceiveThread	Hilo receptor de mensajes, encargado de escuchar, filtrar y encolar los mensajes enviados por los dispositivos cliente.
ServerDiagramThread	Hilo que implementa el protocolo de comunicación C-S, siguiendo el diagrama de estados definido para el servidor. También implementa los mecanismos de validación de clientes.

- **SOASClient:** *Service* que implementa el rol Cliente del sistema. Para reproducir el streaming de vídeo emitido por un servidor hace uso de la clase *RTSPPlayerActivity.*

Atributos Destacados	
Atributo	**Descripción**
state	Estado actual del cliente.
messageQueue	Cola de mensaje recibidos.
lService	Objeto que posibilita la comunicación con el servicio de ubicaciones (LocationService).
lastLocation	Vector desplazamiento más actual del vehículo.
waveLock	Objeto que permite mantener la CPU activa mientras el cliente este en marcha.

Métodos Destacados	
Método	**Descripción**
launchThreads()	Lanza a ejecución los distintos hilos secundarios de los que consta el cliente.
stopThreads()	Interrumpe los hilos secundarios.
startStreaming(String, String)	Permite iniciar la sesión de streaming conectándose al servidor RTSP indicado como argumento (IP, Puerto).
stopStreaming()	Permite finalizar la sesión de streaming desconectándose del servidor RTSP.
getLocation()	Devuelve el vector desplazamiento más actual del vehículo.

Clases Internas	
Clase	**Descripción**
LocationServiceConn	Monitoriza el estado del servicio de ubicaciones (LocationService).
StopPlayerReceiver	Oyente que es notificado cuando el usuario cierra la Activity que reproduce el streaming de vídeo (RTSPPlayerActivity). Su cometido es finalizar la sesión correctamente.
ClientHandler	Oyente encargado de recibir, evaluar y gestionar adecuadamente, los mensajes emitidos desde los hilos secundarios hacia el hilo principal del cliente.
ClientSendThread	Hilo emisor de mensajes.
ClientReceiveThread	Hilo receptor de mensajes, encargado de escuchar, filtrar y encolar los mensajes enviados por los dispositivos servidor.
ClientDiagramThread	Hilo que implementa el protocolo de comunicación C-S, siguiendo el diagrama de estados definido para el cliente. También implementa los mecanismos de validación de servidores.

- **LocationService:** *Service* encargado de proporcionar la ubicación del dispositivo a los *services* cliente (SOASClient) y servidor (SOASServer).

Atributos Destacados	
Atributo	**Descripción**
locationClient	Objeto que actúa como intermediario entre la aplicación y los servicios de localización de Google (*Google Location Services*).
locationRequest	Objeto empleado por locationClient para solicitar la ubicación del dispositivo, con una precisión y una frecuencia de actualización determinadas.
lastLocation	Ubicación más reciente del dispositivo.

Métodos Destacados	
Método	**Descripción**
getLastLocation()	Método público empleado por los clientes del servicio para obtener la localización más actual del dispositivo.
onLocationChanged(Location)	Este método es llamado cuando la ubicación del dispositivo ha cambiado.

Clases Internas	
Clase	**Descripción**
LocalBinder	Clase empleada por los clientes del servicio de ubicación (SOASClient, SOASServer) como interfaz de acceso a sus métodos públicos.

- **SOASMessage:** Clase *serializable* que implementa los mensajes utilizados por el sistema en su protocolo de comunicación C-S. Permite rellenar el contenido de los mensajes, y facilita su emisión y recepción a través de la red.

Atributos Destacados	
Atributo	**Descripción**
type	Tipo de mensaje.
ip	Dirección IP del emisor.
location	Vector desplazamiento del emisor.
speed	Velocidad en Km/h del emisor.
maxResolution	Resolución máxima soportada por el emisor.
rtspPort	Puerto de escucha del servidor RTSP.

Métodos Destacados	
Método	**Descripción**
getType()	Devuelve el tipo de mensaje.
getIp()	Devuelve la dirección IP del emisor.
getLocation()	Devuelve el vector desplazamiento del dispositivo emisor.
getSpeed()	Devuelve la velocidad a la que se desplaza el dispositivo emisor.
getMaxResolution()	Devuelve la máxima resolución de reproducción de vídeo soportada por el dispositivo emisor.
getRTSPPort()	Devuelve el puerto donde escucha el servidor RTSP en el dispositivo emisor.
setType(MessageType)	Permite definir el tipo de mensaje.
setIp(String)	Permite definir la dirección IP del emisor.
setLocation(double[])	Permite definir el vector desplazamiento del dispositivo emisor.
setSpeed(float)	Permite definir la velocidad a la que se desplaza el dispositivo emisor.
setMaxResolution(int[])	Permite definir la máxima resolución de reproducción de vídeo soportada por el dispositivo emisor.
setRTSPPort(int)	Permite definir el puerto donde escucha el servidor RTSP en el dispositivo emisor.

- **SOASMessageQueue:** Clase que implementa una cola de mensajes, la cual garantiza la exclusión mutua en la lectura y en la escritura de mensajes, por parte de los hilos consumidor y productor del cliente y del servidor.

Atributos Destacados	
Atributo	**Descripción**
messageQueue	Lista enlazada donde se almacenan los mensajes.
isEmpty	Indica si la cola está vacía.
isFull	Indica si la cola está llena.

Métodos Destacados	
Método	**Descripción**
takeMessage(long)	Devuelve y elimina el primer mensaje de la cola. Como argumento se ha de indicar el *timeout* máximo de espera.
insertMessage(SOASMessage)	Inserta un mensaje al final de la cola.
clearQueue()	Permite vaciar la cola.

- **StopSOASService:** *Service* que permite detener el sistema de forma correcta.

5.3.3 Contexto

- **AppContext:** Clase que contiene constantes, objetos y métodos comunes a toda la aplicación, de tal manera que éstos puedan ser utilizados por varias clases diferentes con facilidad y rapidez.

Atributos Destacados	
Atributo	**Descripción**
RECEIVE_SERVER_PORT	Constante que almacena el puerto de escucha del servidor del sistema.
RECEIVE_CLIENT_PORT	Constante que almacena el puerto de escucha del cliente del sistema.
toast	Objeto utilizado para mostrar mensajes cortos al usuario.

Métodos Destacados	
Método	**Descripción**
getIPAddress(boolean)	Devuelve la dirección IP de la primera interfaz de red diferente a *localhost*. El argumento permite indicar la versión de IP: True- IPv4, False-IPv6.
getScreenDimensions()	Devuelve las dimensiones (Ancho x Alto) de la pantalla del dispositivo.
serializeMessage(SOASMessage)	Permite serializar un mensaje SOAS, convirtiéndolo en un conjunto de bytes capaces de ser enviados a través de la red.
deserializeMessage(byte[])	Permite reconstruir un mensaje SOAS previamente serializado.
soundState()	Devuelve el estado del sonido del dispositivo.
changeSoundState(int)	Permite modificar el estado del sonido del dispositivo.
getSystemTime()	Devuelve la hora del sistema en el formato HH:mm:ss.SSS
isOnePointVector(double[])	Permite saber si un vector 2D posee dos puntos completamente iguales, y que por tanto tiene módulo nulo.
moveVector(double[], double[])	Permite desplazar un vector 2D a una determinada posición dentro del plano cartesiano.
getDegrees(double[], double[])	Permite obtener el ángulo formado por dos vectores 2D.

5.3.4 Base de Datos

- **SQLiteDatabase:** Clase Android que ofrece métodos para gestionar una base de datos SQLite.

Métodos Destacados	
Método	**Descripción**
openOrCreateDatabase(String, int, CursorFactory)	Permite acceder a una base de datos existente o crear un nueva.
rawQuery(String, String[])	Permite ejecutar una sentencia SQL sobre la base de datos abierta.
execSQL()	Permite ejecutar una sentencia SQL sobre la base de datos abierta, de entre las sentencias que no devuelven datos como resultado.

6. Implementación del sistema

El análisis y el diseño realizados en los capítulos anteriores han permitido definir el funcionamiento y la estructura del sistema. Por lo tanto, el siguiente paso lógico es acometer la construcción propiamente dicha. Así, en este capítulo se exponen los detalles de implementación de los principales elementos que componen el sistema.

6.1 Interfaz Gráfica de Usuario

Como se ha visto en el capítulo anterior, la aplicación desarrollada cuenta con una interfaz gráfica compuesta por cinco pantallas distintas, las cuales son implementadas a través de las *Activities*: *MainActivity*, *CreditsActivity*, *SettingsActivity*, *LogsActivity* y *RTSPPlayerActivity*.

La pantalla principal de la aplicación (*MainActivity*), desempeña la función de mando de control, pues permite activar/desactivar el sistema, y proporciona acceso al resto de las *Activities* (Figura 5.1).

Figura 5.1 Interfaz gráfica pantalla principal.

Por otra parte, la pantalla de créditos (*CreditsActivity*), muestra información breve acerca de la aplicación y el proyecto, como son el nombre de la titulación y de la universidad, el nombre del desarrollador y del tutor, así como la fecha de implementación (Figura 5.2).

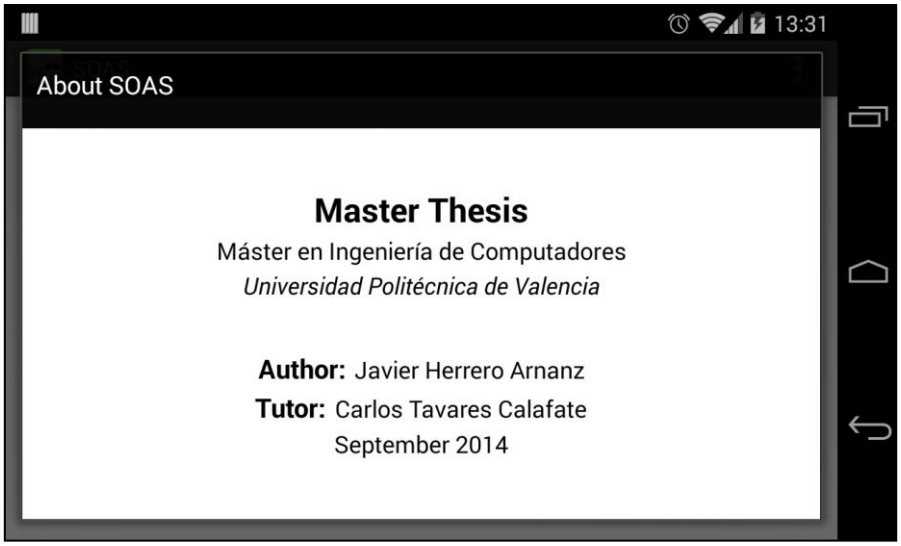


Figura 5.2 Interfaz gráfica pantalla de créditos.

En cuanto a la pantalla de configuración o de ajustes (*SettingsActivity*), ofrece un breve formulario a través del cual se pueden establecer los parámetros de validación utilizados por el sistema (Figura 5.3). El usuario debe guardar de forma persistente, a través del botón habilitado a tal efecto, para que sean aplicados la próxima vez que se ejecute el sistema. La aplicación comprobará que los parámetros introducidos se encuentran dentro del rango válido (0-360), avisando al usuario en caso de que alguno/s de ellos no cumpla/n con dicho requisito. Asimismo, permite restablecer los valores por defecto, y deshabilitar la validación por completo.

Esta pantalla permite realizar pruebas y test del sistema con más facilidad y flexibilidad, posibilitando la búsqueda de los parámetros de validación que más se ajustan a la realidad.

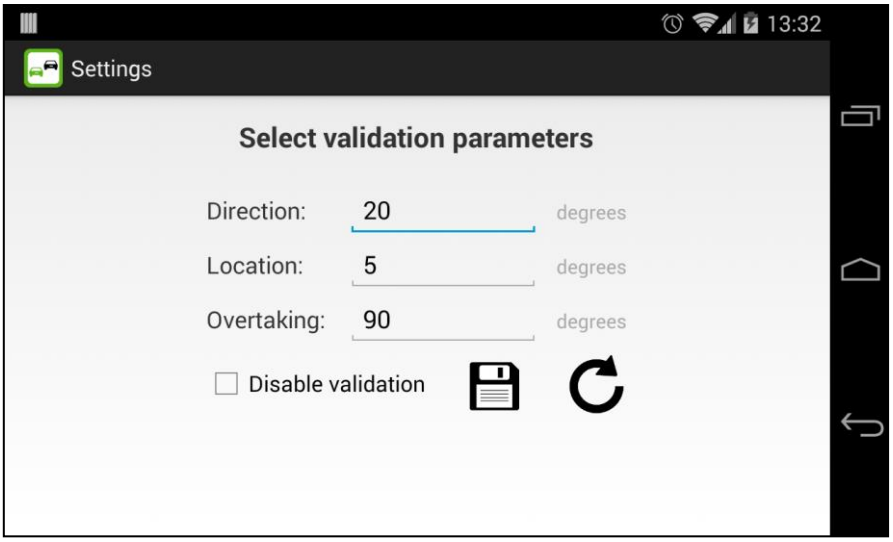


Figura 5.3 Interfaz gráfica pantalla de ajustes.

Otra pantalla que ayuda en las tareas de depuración y en la realización de pruebas, es la pantalla de visualización de logs (*LogsActivity*) (Figura 5.4). A través de ella, el usuario es capaz de repasar los diferentes eventos acontecidos durante la última ejecución del sistema, lo que permite encontrar el origen de posibles comportamientos erróneos. Si lo desea, el usuario también puede eliminar los últimos logs de eventos a través del menú de la *Activity.*

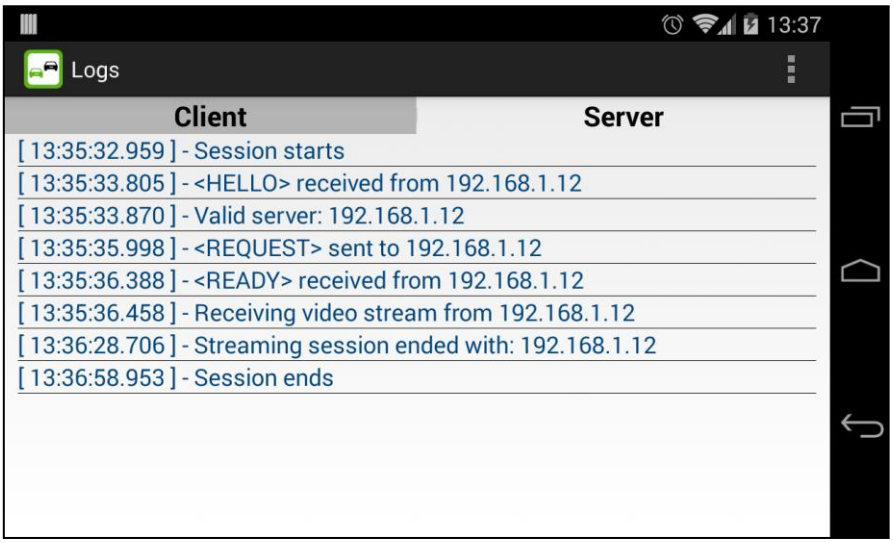


Figura 5.4 Interfaz gráfica pantalla de visualización de logs.

Por otro lado, cuando el usuario activa el sistema, se crea automáticamente una notificación en la barra superior del sistema. Ésta sirve para informar al usuario de la ejecución en segundo plano del sistema, a la vez que permite detenerlo en caso de que sea pulsada (Figuras 5.5 y 5.6).

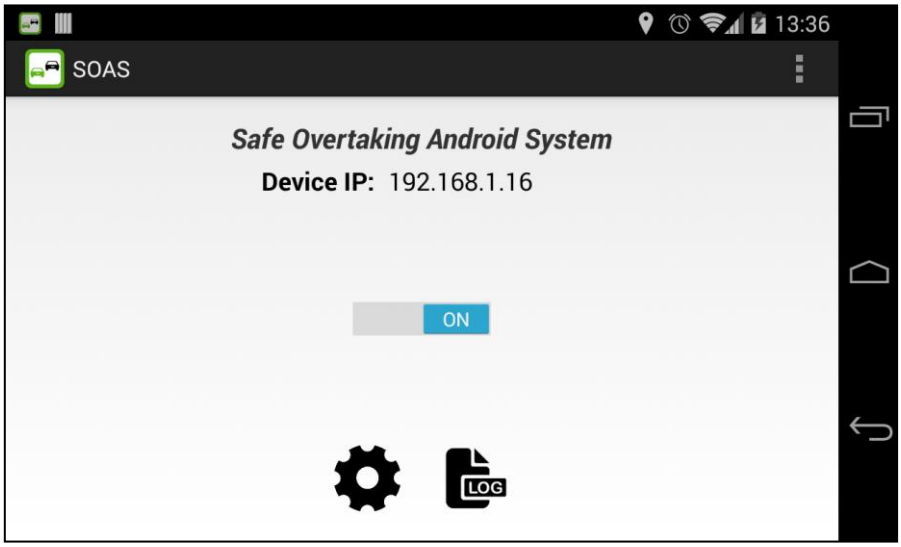


Figura 5.5 Interfaz gráfica pantalla principal con el sistema activado.

Figura 5.6 Notificación para detener el sistema.

La última pantalla, la cual completa la interfaz gráfica de la aplicación, es la pantalla de reproducción de vídeo (*RTSPPlayerActivity*) (Figura 5.7). Ésta se encarga de reproducir el streaming de vídeo RTSP/RTP emitido desde el servidor, mostrándolo con una resolución adaptada a las dimensiones del dispositivo cliente. Además, junto al vídeo, concretamente en la esquina inferior derecha, se muestra un velocímetro con la velocidad actual a la que se desplaza el vehículo servidor. Este valor permite que el conductor del vehículo cliente pueda estimar la aceleración que necesita su vehículo, para completar el adelantamiento en el menor tiempo posible.

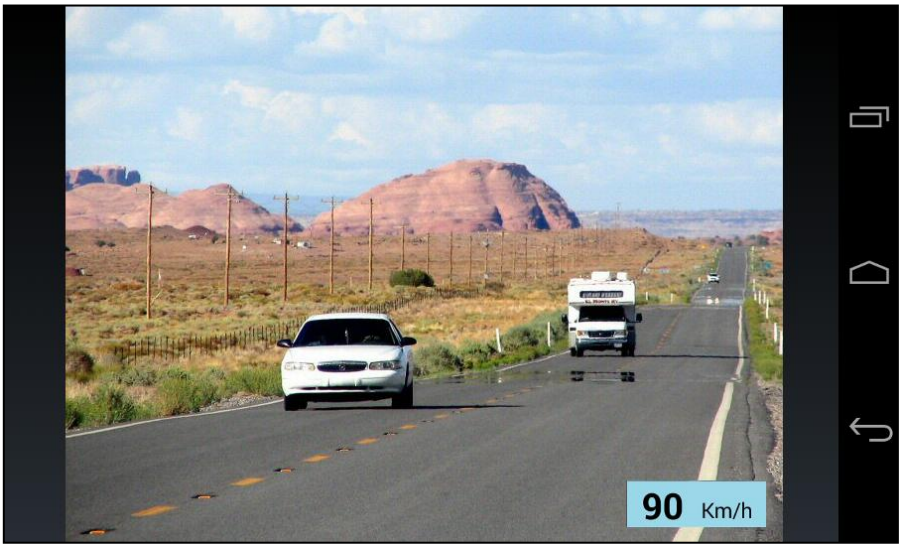


Figura 5.7 Interfaz gráfica pantalla de reproducción.

Por último, en cuanto a la implementación de cada una de estas pantallas, se puede decir que, a grandes rasgos y sin entrar en los detalles más técnicos, todas han sido programadas siguiendo los 3 hitos principales que se exponen a continuación:

1. Definición de la interfaz a través de un *layout* en formato XML (Código 6.1).
2. Implementación de los oyentes para gestionar adecuadamente cada tipo de evento (Código 6.2).
3. Carga de la interfaz, creación y asignación de oyentes al inicializar la *Activity* (Código 6.3).

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/soas_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/SOAS_label"
        android:textSize="20sp"/>

      ...

</RelativeLayout>
```

Código 6.1 Layout MainActivity.

```
private class MainBtnListener implements View.OnClickListener {

            public void onClick(View v) {
                  switch (v.getId()) {
                  case R.id.settings_button: {

                        // Se inicia la actividad de ajustes.

                  }
                  case R.id.logs_button: {

                        // Se inicia la actividad de visualización de logs.

                  }
                  }
            }
}
```

Código 6.2 Oyente MainBtnListener.

```
protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            // Se carga la interfaz de la actividad.
            setContentView(R.layout.activity_main);

            // Se recupera una referencia a los botones y se les asigna un oyente.
            settingsButton = (ImageButton) findViewById(R.id.settings_button);
            logsButton = (ImageButton) findViewById(R.id.logs_button);
            MainBtnListener bListener = new MainBtnListener();
            settingsButton.setOnClickListener(bListener);
            logsButton.setOnClickListener(bListener);

}
```

Código 6.3 Inicialización MainActivity.

6.2 Canal de comunicación

Para lograr la comunicación entre los dispositivos, el sistema hace uso de *sockets* UDP, pues las características del servicio prestado demandan rapidez en la entrega de los datos. La comunicación a través de *sockets* UDP se empleará tanto en el protocolo C-S, como a la hora de servir y recibir el streaming de vídeo. Sin embargo, en esta sección únicamente se trata lo referente al protocolo de comunicación pues, como veremos más adelante, la transmisión y la recepción del streaming de vídeo se apoyan en un par de librerías externas, las cuales requieren ser expuestas en otra sección por separado.

En líneas generales, la puesta a punto de un *socket* UDP, para su uso como emisor o receptor de paquetes UDP, es bastante simple. Bastará con abrir el *socket* UDP en un determinado puerto, emitir el paquete concreto o establecer el *socket* para que escuche paquetes entrantes, y una vez que ya no sea de utilidad cerrar el *socket* (Código 6.4).

```
datagramSocket = new DatagramSocket(port);

datagramSocket.send(packet);

datagramSocket.receive(packet);

datagramSocket.close();
```

Código 6.4 Utilización de Sockets UDP.

Como se puede deducir del fragmento de código anterior, los mensajes del protocolo de comunicación se encapsularán dentro de un paquete UDP. Para que esto sea posible, los objetos que implementan los mensajes del protocolo han de ser *serializables*, de tal manera que puedan ser convertidos en un conjunto de bytes antes de ser empaquetados, y reconstruidos en el destino al ser extraídos.

De este modo, el cliente del sistema (C) necesitará dos *sockets*: uno para emitir mensajes (E), y otro para recibirlos (R). El servidor (S) tendrá los mismos más otro adicional, el cual le permitirá emitir los anuncios (A). Asimismo, puesto que la recepción de paquetes es una operación bloqueante, para lograr concurrencia en la emisión y recepción de mensajes, estos *sockets* son gestionados desde los hilos secundarios del cliente y del servidor. Así, los *sockets* de emisión serán utilizados por los hilos *ClientSendThread* y *ServerSendThread*, los *sockets* de recepción por los hilos *ClientReceiveThread* y *ServerReceiveThread* respectivamente, y por último, el *socket* de emisión de anuncios será gestionado en el hilo del servidor *ServerAdvertiseThread*. La Figura 6.8 muestra un esquema simple del canal de comunicación descrito.

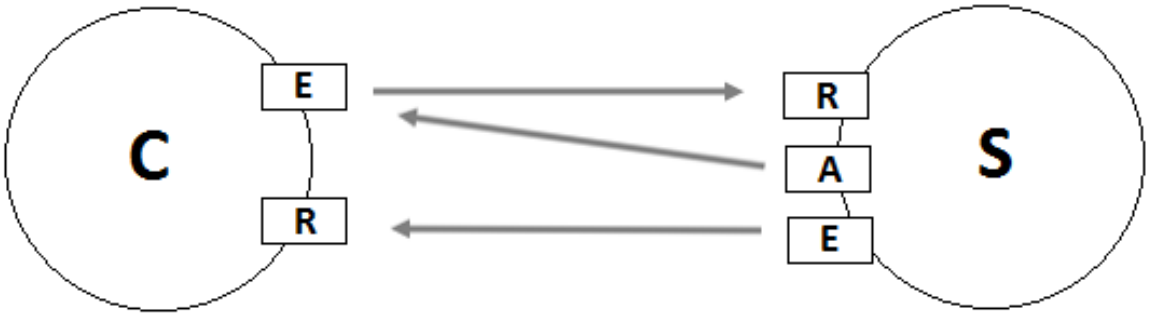


Figura 6.8 Canal de comunicación Cliente-Servidor.

Los hilos de emisión de mensajes, *ClientSendThread* y *ServerSendThread*, serán lanzados a ejecución únicamente cuando sea necesario enviar un mensaje, y se detendrán inmediatamente después de haberlo emitido. En concreto, la petición de emisión de un mensaje tendrá su origen en los hilos donde se implementa el comportamiento del servidor y del cliente, denominados *ServerDiagramThread* y *ClientDiagramThread,* respectivamente.

En lo que respecta al hilo de emisión de anuncios del servidor, *ServerAdvertiseThread,* se encontrará en ejecución desde la activación del sistema hasta su detención. Su frecuencia de envío será de un anuncio por segundo, aproximadamente. Cabe destacar que estos anuncios se enviarán como paquetes *broadcast*, pues deben llegar a todos los clientes cercanos.

Asimismo, los hilos encargados de la recepción de mensajes, *ClientReceiveThread* y *ServerReceiveThread*, también se ejecutarán durante todo el tiempo de vida del *Service*, cliente o

servidor, al que pertenecen. Estos hilos permanecerán bloqueados a la espera de la llegada de un nuevo mensaje. Cuando se produzca dicho evento, analizarán el tipo de mensaje, y si es de utilidad para el estado actual del cliente o del servidor, lo encolaran para su posterior uso. A continuación, volverán a bloquearse a la espera de nuevos mensajes. Este comportamiento queda ilustrado en la Figura 6.9.

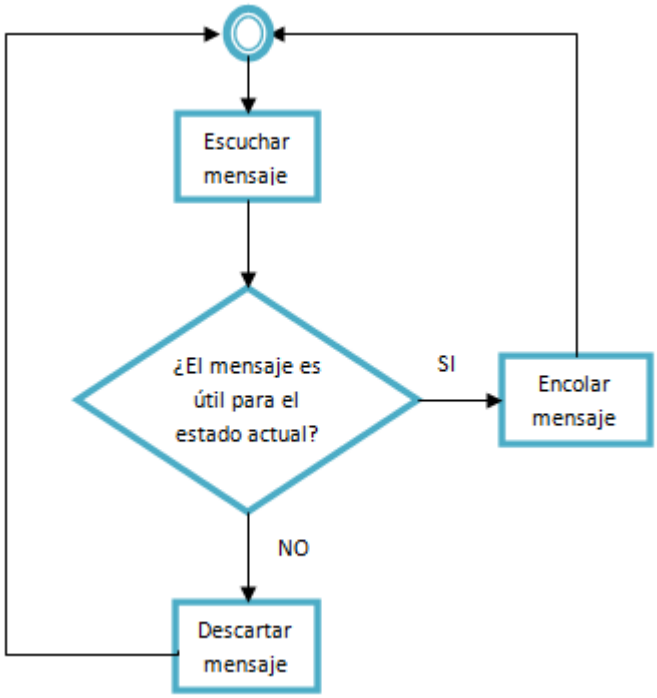


Figura 6.9 Diagrama de flujo hilo receptor de mensajes.

Por tanto, se puede observar que los hilos receptores se comportan como productores de la cola de mensajes del cliente/servidor, mientras que los consumidores serán los hilos *ClientDiagramThread* y *ServerDiagramThread,* respectivamente. Estos hilos implementan los diagramas de estados que definen el protocolo, y los que por ende necesitan la información contenida en los mensajes para operar.

De esta manera, el acceso a la cola de mensajes debe hacerse de forma sincronizada, asegurando la consistencia de los mensajes contenidos en ella. Para ello, la cola se ha implementado como un *monitor*, el cual garantiza la exclusión mutua en la lectura y escritura de mensajes por parte de los hilos consumidor y productor (Códigos 6.5 y 6.6).

```
public synchronized SOASMessage takeMessage(long timeout) {

		while (isEmpty) {
			wait(timeout);
			if ((isEmpty) && (timeout > 0)) { // Venció el timeout.
				return null;
			}
		}

		// Se extrae el mensaje.
		SOASMessage message = messageQueue.poll();

		// Se actualiza el estado de la cola.
		isEmpty = messageQueue.isEmpty();
		isFull = false;

		// Se despierta al productor si se encuentra bloqueado.
		notify();

		// Se devuelve el mensaje al hilo consumidor.
		return (message);
}
```

Código 6.5 Extracción de mensajes.

```
public synchronized void insertMessage(SOASMessage message, long timeout) {

		while (isFull) {
			wait(timeout);
			if ((isFull) && (timeout > 0)) { // Venció el timeout.
				return;
			}
		}

		// Se inserta el mensaje al final de la cola.
		messageQueue.addLast(message);

		// Se actualiza el estado de la cola.
		isEmpty = false;
		if (messageQueue.size() == MAX_SIZE) {
			isFull = true;
		}

		// Se despierta al consumidor si se encuentra bloqueado.
		notify();
}
```

Código 6.6 Inserción de mensajes.

6.3 Geolocalización

Otro elemento esencial del sistema es la obtención de la ubicación del dispositivo, pues como se ha expuesto anteriormente, todo el mecanismo de validación se apoya en los vectores desplazamiento que describen la trayectoria seguida por los vehículos.

Al ser las redes vehiculares un entorno en constante y rápido cambio, es de vital importancia obtener ubicaciones precisas y frecuentemente actualizadas, pues en unos pocos segundos el vector desplazamiento del vehículo puede quedar obsoleto, y por lo tanto ser inservible para el mecanismo de validación. De ahí que se utilice el GPS como proveedor de localizaciones, a una tasa de actualización que oscila entre 1 y 1,5 segundos. Sin embargo, estos parámetros no son controlados en su totalidad por el programador, pues lo que hace éste es solicitar la precisión y la tasa de actualización al servicio de ubicación de Google (*Google Location Service*), y éste hará todo lo posible por satisfacer esos requisitos, pero sin garantizar que se cumplan con exactitud. Así, las ubicaciones recibidas tendrán una precisión variable, y llegarán a una frecuencia lo más cercana posible al intervalo requerido. Por lo tanto, en este punto existe un hándicap que queda fuera del alcance del desarrollador.

En lo que se refiere a la implementación propiamente dicha, tal y como se indicó en el capítulo anterior, existe un *Service* que opera como intermediario entre la aplicación y el proveedor de ubicaciones. Este *Service* es el encargado de suministrar la ubicación más actual al cliente y al servidor. Para implementarlo, en primer lugar, es necesario conectarse al proveedor de ubicaciones y solicitarle actualizaciones periódicas de las ubicaciones del dispositivo (Código 6.7).

```
// Intervalo de actualización.
static final long UPDATE_INTERVAL = 1500; // 1,5 sg.
static final long FASTEST_INTERVAL = 1000; // 1 sg.

// Conexión con el servicio de ubicaciones.
LocationClient  locationClient = new LocationClient(this, this, this);
locationClient.connect();

// Se crea el objeto de petición de ubicaciones y se solicitan.
LocationRequest  locationRequest = LocationRequest.create();
locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
locationRequest.setInterval(UPDATE_INTERVAL);
locationRequest.setFastestInterval(FASTEST_INTERVAL);
locationClient.requestLocationUpdates(locationRequest, this);
```

Código 6.7 Conexión al proveedor de ubicaciones y solicitud de actualizaciones.

A continuación, es preciso definir el método que es invocado cada vez que se recibe una nueva ubicación (Código 6.8).

```
public void onLocationChanged(Location location) {

	// Se actualiza la última ubicación conocida, si la que se acaba de
	// recibir es más actual que la que ya se tenía.
	boolean update;
	if (lastLocation == null) {
		lastLocation = location;
	} else if (location.getTime() > lastLocation.getTime()) {
		lastLocation = location;
	}
}
```

Código 6.8 Actualización de la ubicación.

Asimismo, el cliente y el servidor del sistema recuperarán la última ubicación disponible a través de un método público habilitado a tal efecto (Código 6.9).

```
public Location getLastLocation() {
	return lastLocation;
}
```

Código 6.9 Obtención de la ubicación por parte del cliente y del servidor.

Finalmente, una vez que no se vaya a hacer uso del *Service*, se deben retirar las actualizaciones periódicas y cerrar la conexión con el proveedor de ubicaciones (Código 6.10).

```
public void onDestroy() {

	super.onDestroy();

	// Se eliminan las actualizaciones periódicas.
	locationClient.removeLocationUpdates(this);

	// Se desconecta el LocationClient.
	locationClient.disconnect();
}
```

Código 6.10 Cierre del servicio.

6.4 Protocolo de comunicación

Como se vio en la sección 4.3, además de los diferentes tipos de mensajes, el protocolo de comunicación C-S queda definido a partir de dos elementos más: los diagramas de estados que establecen el comportamiento del cliente y del servidor, y los mecanismos de validación.

Ambos bloques son implementados en los hilos *ClientDiagramThread* y *ServerDiagramThread*, y están construidos siguiendo una misma estructura. A grandes rasgos, dicha estructura consiste en un bucle *while* que se ejecuta de forma constante mientras el hilo no sea interrumpido. Así, dentro de ese bucle, se evalúa el estado actual del cliente o servidor y, en función de cuál sea, se ejecuta un determinado método que encapsula el comportamiento para dicho estado (Código 6.11).

```
// ServerDiagramThread.
while (!thisThread.isInterrupted()){
	switch (state) {
		case NOTIFY:
			doNotify();
			break;
		case REPLY:
			doReply();
			break;
		case STREAM:
			doStream();
			break;
		case END:
			doEnd();
			break;
	}
}
```

```
// ClientDiagramThread.
while (!thisThread.isInterrupted()){
	switch (state) {
		case LISTEN:
			doListen();
			break;
		case REQUEST:
			doRequest();
			break;
		case PLAY:
			doPlay();
			break;
		case END:
			doEnd();
			break;
	}
}
```

Código 6.11 Bucle principal ServerDiagramThread y ClientDiagramThread.

Cada uno de los métodos *do<estado>()* anteriores encapsula una parte de la lógica del protocolo. Es en ellos donde se producen las transiciones entre estados, donde se analizan los mensajes previamente encolados, y desde donde se aplican los mecanismos de validación. A modo de ejemplo, se muestra el método *doNotify()* del servidor (Código 6.12).

En cuanto a los mecanismos de validación, serán encapsulados en dos métodos principales: *isValidServer()* en el caso del cliente, e *isValidClient()* para el servidor. Ambos métodos analizarán la dirección y sentido de la marcha de los vehículos, y su ubicación dentro de la calzada, siguiendo para ello las técnicas ya conocidas. Seguidamente, retornarán un valor *booleano* indicando si el servidor o el cliente examinado es o no optimo para servir o ser servido. La Figura 6.10 muestra el funcionamiento de estos métodos a través de un diagrama de flujo.

```
private void doNotify() {

	boolean requestReceived = false;

	// Se espera la llegada de un mensaje REQUEST.
	do {
		rMessage = messageQueue.takeMessage(0);
		if (rMessage.getType() == MessageType.REQUEST) {
			// Se guarda la IP del cliente.
			clientIP = rMessage.getIp();

			// Se valida el cliente.
			clientOK = isValidClient(rMessage.getLocation());

			// Se cambia de estado.
			synchronized (state) {
				state = ServerState.REPLY;
			}
			requestReceived = true;
		}
	} while ((!requestReceived) && (!thisThread.isInterrupted()));
}
```

Código 6.12 Método doNotify() del servidor.

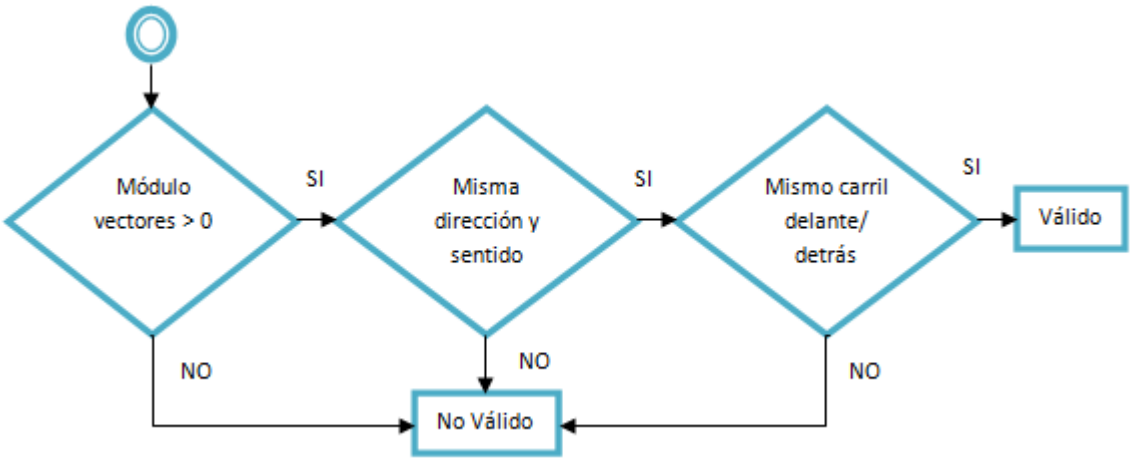


Figura 6.10 Diagrama de flujo del mecanismo de validación.

Por último, indicar que el cliente cuenta con dos métodos adicionales. El primero de ellos es *bestValidServer(candidates)* (Código 6.13), el cual permite seleccionar, de entre los servidores válidos cercanos, a aquel situado inmediatamente delante. Por otro lado está el método *isStreamingUseful()* (Código 6.14), que es utilizado para validar si el adelantamiento se ha completado, en cuyo caso el cliente daría por finalizada la sesión de streaming.

```
private String bestValidServer(HashMap<String, double[]> candidates) {

	String nearbyCandidate = "";
	float minDistance = Float.MAX_VALUE;

	// Se selecciona el candidato más cercano.
	Iterator<String> iterator = candidates.keySet().iterator();
	while (iterator.hasNext()) {

		String cadidateIP = iterator.next();
		double[] candidateL = candidates.get(cadidateIP);
		double[] clientL = getLocation();
		float[] distance = { 9999 };

		// Se calcula la distancia.
		Location.distanceBetween(clientL[2], clientL[3], candidateL[2],
					candidateL[3], distance);

		if (distance[0] < minDistance) {
			// Actualizar candidato más cercano.
			minDistance = distance[0];
			nearbyCandidate = cadidateIP;
		}
	}
	return nearbyCandidate;
}
```

Código 6.13 Método bestValidServer().

```
private boolean isStreamingUseful(double[] serverLoc) {

	// Vector desplazamiento del cliente y vector que une los dos vehículos.
	double[] clientL = getLocation();
	double[] clientToServer = {clientL[2], clientL[3],serverLoc[2], serverLoc[3]};

	// Se calcula el ángulo entre ambos vectores.
	double degrees = AppContext.getDegrees(clientL, clientToServer);
	if (degrees <= valParams[2]) { // Detrás.
		return true;
	} else { // Delante.
		return false;
	}

}
```

Código 6.14 Método isStreamingUseful().

6.5 Transmisión y reproducción de vídeo

Con lo expuesto hasta el momento, cliente y servidor son capaces de establecer comunicación, y decidir si el otro extremo de la conversación está o no involucrado en la maniobra de adelantamiento. Por lo tanto, en caso afirmativo, solo faltaría iniciar la captura y transmisión del streaming de vídeo en el caso del servidor, o lanzar la reproducción si se trata del cliente.

Para lograr implementar esta funcionalidad, no es posible apoyarse únicamente en las herramientas proporcionadas por el API de Android, al menos de momento. Es necesario hacer uso de librerías externas que ya hayan abordado el problema.

De este modo, para el bloque encargado de la captura y transmisión del flujo multimedia, se ha empleado la librería *libStreaming* [24], la cual es capaz de convertir el vídeo capturado por la cámara del dispositivo en un flujo de datos RTP que pueda ser enviado a través de la red. Para el caso de la reproducción ha sido necesaria la ayuda de la librería *libVLC* [22] [23], la cual proporciona un reproductor multimedia capaz de reproducir vídeo RTSP/RTP, entre otros muchos formatos.

A continuación, se describe con más detalle la intervención de estas dos tecnologías en el sistema.

Captura y transmisión

La librería *libStreaming* permite arrancar en el dispositivo un servidor RTSP, el cual emite en tiempo real el vídeo capturado por la cámara al cliente RTSP que lo solicite. Este servidor está implementado a través de un *Service* que opera en segundo plano, concretamente se trata de la clase *RtspServer*, de la cual se habló en el capítulo 5.

Antes de poner en marcha el servidor RTSP, es necesario establecer las características del streaming de vídeo que se va a servir. En concreto se debe indicar si se desea o no incluir la captura de audio, el codec de vídeo a utilizar, la resolución en píxeles, las imágenes o frames por segundo (FPS) a los que se desea que se reproduzca, y el bit rate de codificación en bits por segundo (BPS) (Código 6.15). Igualmente, estos parámetros pueden ser modificados estando el servidor en marcha, pero no cuando se encuentra en medio de una sesión.

```
SessionBuilder
       .getInstance()
       .setAudioEncoder(SessionBuilder.AUDIO_NONE)
       .setVideoEncoder(SessionBuilder.VIDEO_H264)
       .setVideoQuality(new VideoQuality(1280, 720, 30, 2000000));
```

Código 6.15 Inicialización servidor RTSP (1).

El siguiente paso será establecer el puerto de escucha del servidor RTSP, iniciando inmediatamente después su ejecución (Código 6.16).

```
// Se establece el puerto de escucha.
Editor editor = PreferenceManager.getDefaultSharedPreferences(this).edit();
editor.putString(RtspServer.KEY_PORT, "7777");
editor.commit();

// Se arranca el servidor.
startService(new Intent(this, RtspServer.class));
```

Código 6.16 Inicialización servidor RTSP (2).

En este momento, el servidor ya estaría en disposición de recibir un nuevo cliente a quien servir el streaming de vídeo, bajo las características establecidas. A grandes rasgos, el comportamiento interno del servidor al recibir un cliente se puede resumir en los siguientes 3 pasos:

1. Inicia la captura de vídeo a través de un objeto *MediaRecorder.* El cual almacena los datos en el buffer de salida del socket UDP por el que se va a emitir el flujo multimedia.

2. A medida que van llegando los datos al buffer son empaquetados en formato RTP.

3. Se emiten de manera continuada los paquetes RTP generados, a través del socket UDP.

De igual modo, cuando el servidor RTSP ya no sea de utilidad, bastará con finalizar la ejecución del *Service* para detenerlo (Código 6.17).

```
// Se detiene el servidor.
stopService(new Intent(this, RtspServer.class));
```

Código 6.17 Detención del servidor RTSP.

Recepción y reproducción

En cuanto a la librería de reproducción, su utilización también es bastante simple. En general, bastará con crear una instancia de la clase *LibVLC*, y establecer los dos elementos que son esenciales para poder iniciar la reproducción (Código 6.18):

1. La URL del servidor RTSP que sirve el vídeo, la cual será del tipo *rtsp://ip_servidor_rtsp:puerto*

2. La superficie de visualización o *surface,* donde se mostrará el vídeo recibido, y que forma parte de la *Activity RTSPPlayerActivity*.

```
private void createPlayer(String media) {

	// Se crea y arranca el reproductor multimedia.
	libvlc = LibVLC.getInstance();
	libvlc.setHardwareAcceleration(LibVLC.HW_ACCELERATION_DISABLED);
	libvlc.setSubtitlesEncoding("");
	libvlc.setTimeStretching(true);
	libvlc.setChroma("RV32");
	libvlc.setVerboseMode(true);
	libvlc.setNetworkCaching(400);
	LibVLC.restart(this);
	EventHandler.getInstance().addHandler(handler);
	holder.setFormat(PixelFormat.RGBX_8888);
	holder.setKeepScreenOn(true);

	// media = "rtsp://ip_servidor_rtsp:puerto"
	libvlc.playMRL(media);
	libvlc.attachSurface(holder.getSurface(), this);
}
```

Código 6.18 Puesta a punto del reproductor multimedia.

Por último, al igual que sucedía en el caso del servidor RTSP, cuando el reproductor ya no sea necesario deberá ser cerrado correctamente (Código 6.19).

```
private void releasePlayer() {
	if (libvlc == null) {
		return;
	} else {
		EventHandler.getInstance().removeHandler(handler);
		libvlc.stop();
		libvlc.detachSurface();
		libvlc.closeAout();
		libvlc.destroy();
		libvlc = null;
	}
}
```

Código 6.18 Cierre del reproductor multimedia.

6.6 Base de datos

La persistencia de los *logs* de eventos del cliente y del servidor se logra a través de una pequeña base de datos *SQLite*. Una vez que la cantidad de entradas que contendrán los registros es imprevisible, consideramos que esta tecnología es la adecuada para ayudar a trabajar con estos datos de forma simple, rápida y flexible.

La base de datos recibe el nombre de *sessionLogs*, y cuenta con dos tablas de idéntica estructura, las cuales almacenan los logs de eventos del cliente y del servidor, de la última sesión. Dichas tablas se denominan *clientLog* y *serverLog* respectivamente, y contienen los siguientes campos:

- ID - Tipo: Integer - Clave primaria de la tabla.
- TIME - Tipo: Text - Orden temporal del evento.
- INFO - Tipo: Text - Información del evento.

El cliente y el servidor almacenarán los eventos en la tabla que les corresponde, desde el hilo *ClientDiagramThread* y *ServerDiagramThread* respectivamente. Lo primero que tendrán que hacer es limpiar la tabla (Código 6.19).

```
// Apertura base de datos.
SQLiteDatabase db;
db = getApplicationContext().openOrCreateDatabase("sessionLogs",MODE_PRIVATE,null);

// Limpieza Cliente.
db.execSQL("DROP TABLE IF EXISTS clientLog");

// Limpieza Servidor.
db.execSQL("DROP TABLE IF EXISTS serverLog");

// Cierre base de datos.
db.close();
```

Código 6.19 Eliminar contenido de las tablas.

A continuación, durante el transcurso de la sesión, irán añadiendo de forma progresiva la información de los eventos que vayan aconteciendo. En este sentido destacar que cliente y servidor disponen de un método implementado a tal efecto (Código 6.20).

```
private void addInfoToLog(String table, String info) {
	// Apertura.
	SQLiteDatabase db;
	db = context().openOrCreateDatabase("sessionLogs",MODE_PRIVATE, null);

	// Se crea la tabla si no existiera.
	db.execSQL("CREATE TABLE IF NOT EXISTS "+table+" (id INTEGER PRIMARY KEY
		autoincrement, time TEXT NOT NULL, info TEXT NOT NULL)");

	// Se inserta la información del evento.
	db.execSQL("INSERT INTO "+table+" VALUES(null,'"+AppContext.getSystemTime()+"'
		"+info+"')");

	// Cierre.
	db.close();
}
```

Código 6.20 Insertar información de un evento.

Por último, solo queda mostrar al usuario la información recopilada anteriormente. Este proceso se lleva a cabo en la *Activity LogsActivity*, y básicamente consiste en obtener todo el contenido de la tabla cliente o la tabla servidor, e ir imprimiendo en pantalla una a una cada línea (Código 6.21).

```
// Apertura.
SQLiteDatabase db = this.openOrCreateDatabase("sessionLogs", MODE_PRIVATE, null);

// Se obtienen los registros.
Cursor cursor = db.rawQuery("SELECT * FROM " + tableBD + " ORDER BY time ASC",
       new String[] {});

// Se muestran en la Activity en forma de tabla.
while (cursor.moveToNext()){

       ...

       row_content.setText(cursor.getString(1) + " - " + cursor.getString(2));
       row.addView(row_content);
       table.addView(row);
       table.addView(line);
}

// Cierre.
cursor.close();
db.close();
```

Código 6.21 Lectura e impresión de la información.

7. Pruebas

Una vez analizado, diseñado e implementado el sistema, solo falta probar que su funcionamiento es el que cabía esperar. Así, en este capítulo se exponen las diferentes pruebas realizadas, los escenarios donde se han llevado a cabo, y los resultados obtenidos. Todo este proceso permitirá evaluar en qué grado el sistema está cumpliendo con los requisitos y objetivos marcados.

Las pruebas aquí descritas se han realizado en un entorno simulado. Concretamente, los dispositivos reciben su ubicación a través de coordenadas almacenadas en un fichero de texto, las cuales describen una ruta previamente fijada. Además, se comunican a través de una red LAN Wi-Fi en modo infraestructura. Se decidió seguir este procedimiento debido a que facilita en gran medida la ejecución de las pruebas y la obtención de resultados. Es posible realizar varias iteraciones de un mismo test en poco tiempo, y la medición y análisis de los datos puede llevarse a cabo de forma más rápida. Si bien es cierto que, en un entorno simulado, los valores obtenidos son menos precisos que los que se hubiera podido obtener en un escenario real, consideramos que igualmente permiten estudiar el comportamiento del sistema con total claridad.

7.1 Dispositivos móviles utilizados

Para la realización de la pruebas se han utilizado varios dispositivos móviles con características diferentes, lo que permite obtener una mejor perspectiva acerca del comportamiento del sistema. En concreto se han empleado 3 dispositivos móviles, 2 de ellos teléfonos, más una tablet. Las especificaciones técnicas de estos dispositivos que pueden tener un mayor interés, o que pueden suponer un mayor impacto sobre el sistema, son las siguientes (Tabla 7.1):

Dispositivo	SO	CPU	RAM	Pantalla	Vídeo Cámara
ASUS Memo Pad HD 7	Android 4.2.2	Quad-Core 1.2 GHz	1GB	1280x800 px	1920x1080 px
LG Nexus 4	Android 4.4.4	Quad-Core 1.5GHz	2GB	1280x768 px	1920x1080 px
Samsung Galaxy Ace	Android 2.3.6	832MHz	278MB	480x320 px	640x480 px

Tabla 7.1 Especificaciones técnicas de los dispositivos.

7.2 Ajuste de los parámetros de validación

En primer lugar es necesario comprobar si la implementación de los mecanismos de validación es correcta, siendo capaz el sistema de discriminar si un determinado vehículo es o no optimo para servir o ser servido. Para tal cometido, será preciso ajustar los parámetros de validación a aquellos valores que reflejen más fielmente la realidad.

El procedimiento seguido define 11 escenarios diferentes, los cuales pretenden cubrir las condiciones más usuales que se pueden presentar en una vía de circulación, y en las que por tanto deberá saber desenvolverse el sistema. Las características de dichos escenarios se muestran en la Tabla 7.2.

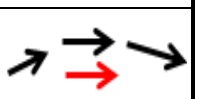
Nº Vehículos	Dirección	Sentido	Carril	Otros	Figura
2	Idéntica	Idéntico	Mismo carril	Un vehículo delante del otro	
2	Ligeramente distinta	Idéntico	Mismo carril	Un vehículo delante del otro	
2	Perpendicular	Opuesto	Carriles en una intersección	-	
2	Idéntica	Opuesto	Carriles paralelos	-	
2	Idéntica	Idéntico	Carriles paralelos	Vehículos a la misma altura	
2	Idéntica	Idéntico	Carriles paralelos	Un vehículo más adelantado que el otro	
2	Distinta	Idéntico	Mismo carril	El primer vehículo se encuentra dentro de una curva	
3	Idéntica	Idéntico	Mismo carril	Vehículos formado una fila	
3	Idéntica	Idéntico	Dos vehículos en un mismo carril, el otro en el carril paralelo	-	
3	Idéntica	Idéntico y Opuesto	Dos vehículos en un mismo carril, el otro en el carril paralelo	El vehículo del carril paralelo circula en sentido opuesto	
2	Idéntica	Idéntico	Mismo carril - Carril paralelo - Mismo carril	El segundo vehículo adelanta al primero	

Tabla 7.2 Características de los escenarios de pruebas.

Para aplicar cada uno de los escenarios anteriores al sistema, se modelaron sus características a través de diferentes trayectorias que encapsulaban las condiciones marcadas por cada uno. Debían ser lo suficientemente largas como para garantizar la duración de los escenarios durante un periodo de tiempo que permitiera observar y validar el comportamiento del sistema.

Cada escenario se ejecutó en el sistema varias veces, modificando los parámetros de validación en cada una de las iteraciones, hasta lograr ajustarlos en aquellos valores que no generasen falsos positivos o falsos negativos. Los umbrales de validación finales pueden observarse en la Figura 7.1.

Parámetro	Valor
Umbral dirección-sentido	20°
Umbral ubicación	5°
Umbral adelantamiento	90°

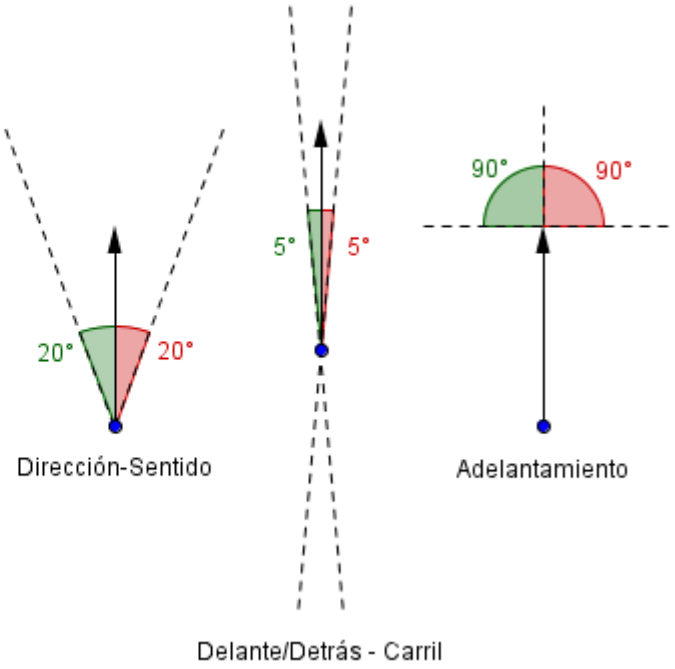


Figura 7.1 Parámetros de validación.

Para el umbral de dirección y sentido se estipuló un ángulo de 20°, pues al mismo tiempo que permite discernir aquellos vehículos que van en sentido contrario, también permite evaluar positivamente aquellas situaciones en las que dos vehículos en el mismo carril llevan una dirección significativamente distinta. Este caso puede darse, por ejemplo, cuando el primero de los vehículos se encuentra en la entrada de una curva, mientras que el que lo sigue aun se encuentra en la recta previa.

Por otra parte, el umbral para evaluar la ubicación tiene un valor pequeño, 5°, para evitar falsos positivos provocados por vehículos situados en los carriles paralelos al que se está evaluando.

Por último, el umbral para detectar si el adelantamiento ya se ha completado, es el más rápido de fijar de todos pues, en el instante en que los vehículos se encuentran completamente en paralelo, el vector desplazamiento del cliente y el vector que une a cliente con servidor, formaran 90° exactos, con lo que un ángulo superior a ese valor indica que el cliente ya se encuentra por delante del servidor.

7.3 Escenario ideal vs Escenario no ideal

Todos los escenarios utilizados anteriormente para ajustar los umbrales de validación son escenarios ideales, es decir, los vehículos obtienen ubicaciones completamente precisas, y a la frecuencia de actualización estipulada. Sin embargo, como ya se comentó en el capítulo anterior, los servicios de geolocalización reales devuelven ubicaciones aproximadas, y no siempre en el instante que se esperan. Por lo tanto, es necesario comprobar cómo se comportaría el sistema bajo esas condiciones no ideales.

Para tal cometido, se establecerá un escenario concreto y se evaluará el comportamiento del sistema bajo condiciones ideales. A continuación, se añadirá cierta aleatoriedad en la precisión y tiempo de llegada de las ubicaciones, y se volverá a ejecutar el sistema.

Para evaluar el comportamiento se medirán tres parámetros concretos:

- Falsos positivos: Número de veces que un dispositivo, cliente o servidor, da por válido a un servidor o cliente que no lo es.
- Falsos negativos: Número de veces que un dispositivo, cliente o servidor, rechaza a un servidor o un cliente válido.
- Tiempo de conexión: Tiempo transcurrido desde que un cliente recibe un anuncio desde un servidor válido, hasta que finalmente se conecta a él y empieza a recibir el streaming de vídeo.

Asimismo, la medición de los parámetros se repetirá 10 veces para cada uno de los dos escenarios, de tal manera que los valores finales se obtendrán a partir de la media aritmética.

El escenario estará formado por 3 vehículos, los cuales circulan uno detrás del otro, a corta distancia, y siguiendo la trayectoria definida en la Figura 7.2. De ese modo, durante todo el trayecto, el primero de los vehículos se comportará como servidor del segundo. El segundo como cliente del primero y servidor del tercero. Y en el caso del vehículo situado en tercer lugar, será el cliente del segundo vehículo.

Por último, indicar que en el escenario ideal, las ubicaciones tendrán precisión absoluta, mientras que en el escenario no ideal, podrán ser aleatoriamente desviadas dentro de una circunferencia de 10 metros de radio. Por otra parte, en el escenario ideal las ubicaciones llegarán a una frecuencia de 1,5 segundos, mientras que en el otro escenario, este tiempo podrá verse incrementado hasta 2 segundos más.

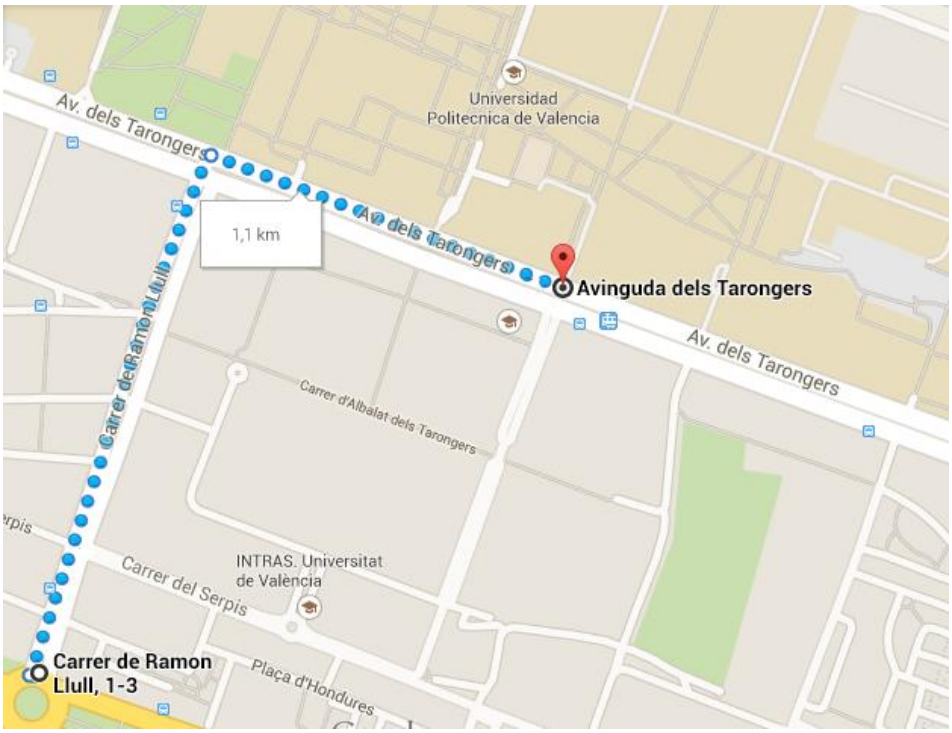


Figura 7.2 Recorrido simulado.

Resultados

A continuación se muestran los resultados obtenidos en la simulación, una vez aplicada la media aritmética a cada conjunto de valores (Tablas 7.3 y 7.4)

ESCENARIO IDEAL	
Tiempo de conexión	4,3 seg
Falsos positivos	0
Falsos negativos	0

Tabla 7.3 Resultados escenario ideal.

ESCENARIO NO IDEAL	
Tiempo de conexión	4,08 seg
Falsos positivos	5,8
Falsos negativos	9,2

Tabla 7.4 Resultados escenario no ideal.

En primer lugar, los resultados obtenidos para el escenario ideal confirman el correcto funcionamiento del sistema. Por un lado se observa que el tiempo de conexión se encuentra en un rango aceptable, pues si este parámetro fuese mucho mayor, es posible que en el instante en el que se estableciera la conexión, el streaming de vídeo ya no fuese de utilidad para el conductor. Por otra parte, el que no se hayan

producido falsos positivos o falso negativos demuestra que los mecanismos de validación discriminan correctamente a clientes y servidores. Esto es debido a que, al recibir ubicaciones precisas y actualizadas, los vectores desplazamiento reflejan fielmente la trayectoria seguida por los vehículos.

Para el segundo de los escenarios, el comportamiento del sistema se tornó aleatorio, pues las ubicaciones recibidas por los dispositivos no permiten expresar el desplazamiento real de los vehículos. A causa de esto, los mecanismos de validación cometen errores que pueden ser falsos positivos o falsos negativos, sin seguir una distribución concreta. Asimismo, se comprobó que, al relajar los umbrales de validación, se cometían menos errores. Sin embargo, esta no es una solución válida, pues en un entorno real, con un mayor número de vehículos, a la larga propiciaría cometer mayor número de equivocaciones.

En lo que respecta al tiempo de conexión, no se obtuvieron novedades con respecto al primer escenario. En aquellas interacciones en las que se logró conectar dos dispositivos consecutivos, el tiempo empleado se mantuvo de nuevo en torno a los 4 segundos.

La conclusión final que se puede extraer a tenor de estos resultados es que el funcionamiento del sistema, así como los mecanismos de validación propuestos, son óptimos bajo un escenario donde los servicios de geolocalización proporcionen ubicaciones con errores reducidos.

7.4 Rendimiento

Para completar la validación del sistema, al mismo tiempo que se ejecutaban las pruebas de la sección anterior, se midió el rendimiento del sistema en cada uno de los tres dispositivos utilizados. Para ello, a través del IDE Eclipse, se midió la tasa de utilización de CPU cuando el dispositivo analizado se comportaba como cliente, como servidor, o como cliente y servidor al mismo tiempo. Los valores finales de este parámetro, tras aplicar la media aritmética, se pueden observar en la Tabla 7.5.

	ROL		
	Cliente	**Servidor**	**Cliente-Servidor**
ASUS Memo Pad HD 7	43%	7%	54%
LG Nexus 4	29%	8,5%	26%
Samsung Galaxy Ace	57%	3,4%	37%

Tabla 7.5 Tasa de utilización de CPU.

Como puede observarse en los valores anteriores, el reproductor multimedia (cliente) es el módulo que mas sobrecarga produce, mientras que la captura y emisión del streaming de vídeo (servidor) tiene un impacto considerablemente menor.

Destaca el valor obtenido para el dispositivo *Samsung Galaxy Ace*, al operar como cliente y servidor al mismo tiempo. El uso de CPU en este escenario es menor que el alcanzado cuando trabaja únicamente como cliente. Este hecho puede deberse a que el sistema Android, a causa de una alta utilización de la CPU, finalizara la ejecución de algunos de los componentes del reproductor multimedia pues el dispositivo era capaz de emitir el streaming de vídeo sin problemas, pero en ningún momento llego a reproducir el flujo multimedia entrante. Con lo cual, se puede concluir que la correcta ejecución de la aplicación está garantizada en los dispositivos móviles actuales de media y alta gama. Destacar también que, aunque el uso de CPU al operar como cliente es importante, el sistema se ejecuta en un contexto en el cual el usuario no estará haciendo un uso exhaustivo del dispositivo, pues va conduciendo.

8. Conclusiones

Tras la elaboración de este trabajo, se puede decir que se ha alcanzado el objetivo principal marcado al inicio, pues el sistema desarrollado es capaz de transmitir vídeo en tiempo real, logrando así asistir a los conductores en las maniobras de adelantamiento. Asimismo, se ha conseguido que sea un sistema selectivo, pues el cliente sólo recibirá el vídeo del vehículo situado inmediatamente delante de él, mientras que el servidor sólo se lo enviará al vehículo situado inmediatamente detrás.

No obstante, existe una serie de limitaciones que no han permitido alcanzar la exactitud deseada en determinados aspectos del sistema. Con la intención de mejorarlos y de proporcionar un mejor servicio, se han propuesto ciertas tareas complementarias que pueden ser desarrolladas en el futuro próximo.

8.1 Limitaciones

A pesar del correcto funcionamiento de la aplicación, tal y como se ha comentado, hay una serie de restricciones técnicas que han dificultado la consecución más precisa de los objetivos o planteamientos de implementación.

A continuación se presentan los principales aspectos que han dificultado un desarrollo más exacto:

- El **sistema de geolocalización de Android** es **poco preciso** para los términos en los que se desenvuelve el sistema, pues una desviación de unos poco metros en las coordenadas proporcionadas genera errores insalvables en los mecanismos de validación.

- La **tecnología Ad-Hoc** en Android es **poco accesible**, pues no puede ser utilizada de forma directa, lo que dificulta la puesta en marcha del sistema en el exterior.

- **Android no facilita la difusión y reproducción de vídeo en tiempo real** a través de su API. Esto supone una mayor carga de trabajo para el desarrollador, pues debe implementarlo por si solo, o buscar otras herramientas que satisfagan sus necesidades y encajen con el sistema ha desarrollar.

- **Dificultad para encontrar un sistema funcional en todos los dispositivos**, puesto que existen demasiados condicionantes tecnológicos que obstaculizan su validez. Entre éstos se incluyen la versión del SO, la arquitectura de la CPU y los códecs multimedia disponibles, entre otros.

8.2 Trabajos futuros

En esta sección se realizan algunas propuestas para mejorar las limitaciones anteriores, así como otras para ampliar el desarrollo y la utilidad del sistema.

- Sustituir los mecanismos de validación basados en vectores de desplazamiento por otros que se apoyen en el **análisis de la matrícula de los vehículos**, a través de técnicas de realidad aumentada y tratamiento de imágenes. Concretamente, consistiría en que los vehículos incluyeran sus matriculas en los mensajes del protocolo, de tal manera que los vehículos receptores de dichos mensajes pudieran analizar si dicha matrícula se corresponde con la del vehículo situado delante suyo. Este nuevo mecanismo eliminaría los problemas ocasionados por la falta de precisión en la obtención de ubicaciones geográficas.

- Aplicar técnicas que permitieran **evaluar las condiciones del canal físico de comunicación**, de modo que fuera posible adecuar las características del vídeo, con el fin de lograr un mayor aprovechamiento del ancho de banda del canal, sin llegar a saturarlo.

- **Mejorar la seguridad del sistema**, mediante la implementación de técnicas de cifrado y codificación de la información, o a través de la implantación de mecanismos para la prevención y detección de ataques, producidos a través del protocolo de comunicación C-S.

- **Incluir consejos y normas de seguridad vial**, dado que el sistema que se ha desarrollado puede servir para captar la atención de los ciudadanos en el problema de la seguridad en las vías de circulación. Con técnicas obsoletas el ciudadano no presta demasiada atención, pero con las nuevas tecnologías hay más oportunidad para transmitirles determinada información.

Tras detallar los posibles trabajos que podrían llevarse a cabo, cabe añadir que si se pretendiese expandir este proyecto a un ámbito más profesional, sería necesario evaluarlo tratando de estimar si los beneficios serían superiores a los costes. Por tanto, en un futuro próximo, además de incluir las mejoras propuestas, sería interesante elaborar un Plan de Marketing con la ayuda de un experto.

En último término, citar las aportaciones de este trabajo en el ámbito personal y en el laboral. En el terreno personal me ha permitido enfrentarme a nuevos retos, al llevar a cabo el desarrollo del sistema en un campo de estudio nuevo para mí, las redes vehiculares. Además, se trata de una aplicación que hace uso de dos librerías completamente independientes, cuya adhesión al sistema ha supuesto todo un desafío. En un terreno más profesional, considero que puede ofrecerme oportunidades laborales en el futuro, puesto que el sistema ha sido desarrollado para dispositivos móviles, cuyo mercado está actualmente en auge.

9. Bibliografía

[1] S. Zeadally, R. Hunt, Y.-S. Chen, A. Irwin, y A. Hassan (2012), "Vehicular ad hoc networks (VANETS): status, results,and challenges", en Telecommunication Systems, vol.50, no.4, pp.217-241.

[2] H. Hartenstein, y K. P. Laberteaux (2010), "VANET - Vehicular Applications and Inter-Networking Technologies", Editorial: Wiley, ISBN: 9780470740569.

[3] D. Johnson, D. Maltz, y Y.-C. Hu (2007), "The dynamic source routing protocol (DSR) for mobile ad hoc networks for IPv4", vol.260.

[4] C. Perkins, E. Belding-Royer, y S. Das (2003), "Ad hoc on-demand distance vector (AODV) routing".

[5] T. Clausen et al. (2003), "Optimized link state routing protocol (OLSR)".

[6] Plan Mundial para el Decenio de Acción para la Seguridad Vial 2011-2020. (2010). Online: http://www.who.int/roadsafety/decade_of_action/plan/plan_spanish.pdf

[7] Hacia la Sociedad de la Información y el Conocimiento. Capítulo 6. (2010). Online: http://www.prosic.ucr.ac.cr/sites/default/files/documentos/capitulo_06_4.pdf

[8] P. Papadimitratos, A. La Fortelle, K. Evenssen, R. Brignolo, y S. Cosenza (2009), "Vehicular communication systems: Enabling technologies, applications, and future outlook on intelligent transportation", en Communications Magazine, IEEE , vol.47, no.11, pp.84-95.

[9] J. Zaldivar, C. T. Calafate, J.-C. Cano, y P. Manzoni (2011), "Providing accident detection in vehicular networks through OBD-II devices and Android-based smartphones", en Local Computer Networks (LCN), 2011, IEEE 36th Conference, pp.813-819.

[10] J. E. Meseguer, C. T. Calafate, J.-C. Cano, y P. Manzoni (2013), "Characterizing the Driving Style Behavior using Artificial Intelligence Techniques", en Local Computer Networks (LCN), 2013, IEEE 38th Conference.

[11] S. Diewald, A. Möller, L. Roalter, y M. Kranz (2012), "DriveAssist – A V2X-Based Driver Assistance System for Android", en Mensch & Computer Workshopband, pp. 373-380.

[12] B. Liu, D. Ghosal, Y. Dong, C. N. Chuah, y M. Zhang (2011), "CarbonRecorder: A Mobile-Social Vehicular Carbon Emission Tracking Application Suite", en Vehicular Technology Conference (VTC Fall), 2011 IEEE, pp.1-2.

[13] P. Gomes, F. Vieira, y M. Ferreira (2012), "The See-Through System: From implementation to test-drive", en Vehicular Networking Conference (VNC), 2012 IEEE , pp.40,47.

[14] E. Burnette (2011), "Android (Programación)", Editorial: ANAYA Multimedia, ISBN: 9788441528765.

[15] AllJoyn Framework. Online: https://www.alljoyn.org/

[16] Wi-Fi P2P. Online: http://developer.android.com/guide/topics/connectivity/wifip2p.html

[17] Location Strategies. Online: http://developer.android.com/guide/topics/location/strategies.html

[18] J. F. Kurose, y K. W. Ross (2010), "Redes de computadoras: Un enfoque descendente", Editorial: Addison-Wesley, ISBN: 9788478291199.

[19] I. Sommerville (2005), "Ingeniería del software", Editorial: Addison-Wesley, ISBN: 9788478290741.

[20] C. Larman (2005), "Applying UML and Patterns: an introduction to object-oriented analysis and design and iterative development", Edición: 3ª , Editorial: Prentice-Hall, ISBN: 9780131489066.

[21] J. Arlow, y I. Neustadt (2006), "UML 2 (Programación) ", Editorial: Anaya Multimedia, ISBN: 9788441520332.

[22] LibVLC. Online: http://www.videolan.org/vlc/libvlc.html

[23] LibVLC para Android. Online: https://wiki.videolan.org/AndroidCompile

[24] LibStreaming. Online: https://github.com/fyhertz/libstreaming