Universidad Carlos III de Madrid

Escuela Politécnica Superior

Departamento de Ingeniería Telemática

PROYECTO FIN DE CARRERA

Ingeniería de Telecomunicación

ESTUDIO DEL SIMULADOR DE REDES VEHICULARES VEINS

PABLO GONZÁLEZ-RIPOLL CEREZO

Tutorizado por Estrella García Lozano

Madrid, julio de 2012

Título: ESTUDIO DEL SIMULADOR DE REDES VEHICULARES VEINS
Autor: Pablo González-Ripoll Cerezo
Tutora: Estrella García Lozano

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 18 de julio de 2012 en Leganés (Madrid), en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

"No es que no nos atrevamos porque las cosas sean difíciles, simplemente las hacemos difíciles cuando no nos atrevemos"

Lucio Anneo Séneca

Agradecimientos

No sé si con los años soy por fin capaz de ver mi paso por la carrera con perspectiva, o más bien, ya la he perdido por completo. Sí, conmigo mismo soy de aplicar poco el razonamiento lógico ingenieril, quizás sea por aquello de *en casa del herrero, cuchillo de palo*.

Ahora lo único que acierto a plantear, lo único que consigo preguntarme, es ¿cómo he llegado hasta aquí? Y para eso sí tengo la respuesta clara.

Gracias a toda mi familia, los que están y los que se fueron, especialmente a mi madre, mi padre y mi hermana, por ser el ejemplo a seguir y el apoyo; gracias a mi tío, por hacer de hermano mayor en muchas ocasiones; gracias a tanta gente que dejé en Córdoba (y a los que se vinieron conmigo), en especial a Luis, mi compañero de fatigas; gracias a mi familia madrileña (mis hermanos Félix y Alex los primeros) y mil gracias a Ana, por darme las fuerzas para seguir adelante y sobre todo, por aguantarme.

Mi agradecimiento a Estrella García, por darme la oportunidad de adentrarme en el mundo vehicular y ayudarme a encontrar el sentido a mis largas horas de trabajo.

Resumen

El presente Proyecto Fin de Carrera presenta al lector una incursión en profundidad en el mundo de las redes vehiculares (VANETs), las analiza conceptualmente y en términos de requerimientos y muestra su potencial en lo referente a su amplia gama de aplicaciones, enfocadas a la seguridad vial, entre otras.

Se concluirá que las herramientas de simulación son un medio de vital importancia para su desarrollo, pues permiten estudiar, hacer una valoración de las prestaciones y, en definitiva, verificar la viabilidad de los sistemas de comunicación entre vehículos. Todo ello sin la necesidad de importantes inversiones en experimentos de campo o nuevas infraestructuras. Se verá, por tanto, que el trabajo enfocado al estudio y desarrollo de estas herramientas no es más que una apuesta por el futuro de las propias redes vehiculares.

Esto llevará a un análisis del estado del arte de las herramientas de simulación para estas redes, prestando una especial atención en los sistemas de código abierto.

Nos centraremos en el marco de simulación VEINS, por tratarse de una herramienta que ofrece muy buenas prestaciones y aporta un valor añadido sobre otras soluciones de simulación híbridas similares: el acoplamiento bidireccional entre la simulación de red y la simulación de tráfico.

A partir de este punto, el trabajo irá enfocado a analizar los paradigmas básicos de cada uno de los componentes de VEINS (OMNeT++, SUMO y el módulo TraCI, esencialmente) y aportar un estudio sobre las funcionalidades que ofrecen. Para finalizar, se llevará al terreno práctico lo explicado, de forma que desarrollará una guía para instalación y puesta en marcha de todos los módulos que conforman VEINS y se darán las directrices para llevar a cabo simulaciones con este sistema.

Palabras clave: Redes vehiculares, Simuladores de VANETs, VEINS, SUMO, OMNeT++, sistema TraCI.

Abstract

This Final Project presents to the reader a deep incursion into the world of vehicular ad-hoc networks (VANETs). They will be conceptually analyzed, studied in terms of requirements, and eventually, their high potential as regards their applications (focused on road safety, among others) will be shown.

As a result, a conclusion will arise, that the simulation tools are of vital importance to their development. They allow study, performance assessment and ultimately test the feasibility of inter-vehicles communication systems. All with no need for major investments in field experiments or new infrastructure. It will be seen therefore that the work focused on the study and development of these tools is just a bet on the future of vehicular networks themselves.

In that context, the state of the art of VANETs simulation systems will be analyzed, paying attention to the open source ones.

The main focus will be set on the VEINS simulation framework, due to its high performance results and because of its unique feature: a bidirectional coupling between the network and the traffic simulators.

From that point on, the work will be aimed at analyzing the basic paradigms of all VEINS components (OMNeT++, SUMO and the TraCI module) and providing a study on their features.

To conclude, everything previously explained will be put into practice. An installation and packages building guide will be developed, and the steps in order to run a simulation will be made clear, based on examples.

Keywords: Vehicular ad-hoc networks, VANETs Simulators, VEINS, SUMO, OMNeT++, TraCI system.

Índice general

Índice de figuras

Capítulo 1

Introducción y objetivos

1.1 Planteamiento del problema

El tráfico urbano e interurbano representa uno de los hitos más problemáticos de la sociedad contemporánea. No en vano, las cifras de mortalidad nos abruman a día tras día. Los accidentes de tráfico motivados por fallos de los conductores suelen ir asociados al consumo de alcohol, al estrés o distracciones debidas a diversas causas, principalmente al uso del teléfono móvil. Por otro lado, las congestiones de tráfico son un mal endémico de las grandes y medianas ciudades, lo que ocasiona elevados niveles de contaminación, así como un incremento en el tiempo que se dedica a recorridos habituales, a la vez que en su coste.

Claramente, la calidad de vida de las sociedades industrializadas modernas se está viendo degradada por todos estos efectos.

En los últimos años se han multiplicado los esfuerzos de la comunidad científica e investigadora dedicados a mitigar estos problemas derivados del tráfico vial. Múltiples soluciones se han propuesto, algunas de las cuales han llegado a verse implementadas, tales como recepción de información relativa al estado de las carreteras a través de

señales de radio, señalización de puntos peligrosos mediante elementos dinámicos (carteles que anuncian problemas puntuales), sistemas de telepeaje, asistentes a la conducción y sistemas de seguridad e infraestructuras de última generación, etc.

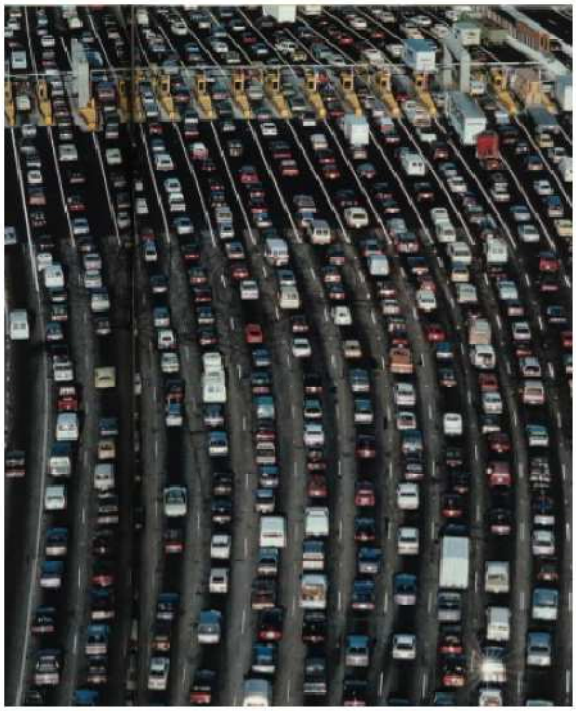


Figura 1. Nuevas soluciones emergen para optimizar el tráfico sin cambiar la topología de red [StreetW]

Los sistemas de control a bordo, que permiten mantener al conductor informado del estado real del vehículo en todo momento, los mecanismos de seguridad pasiva de los que disponen la mayoría de los automóviles (frenos *ABS*, etc.), sistemas de posicionamiento y asistencia a la navegación (*GPS*) o sensores para ayuda al aparcamiento, todos ellos suponen importantes avances tecnológicos cuya finalidad es luchar contra focos de riesgo potencial y hacer más cómoda y eficiente la conducción, en términos de tiempo, dinero y contaminación. Pero serán los avances en el campo de las comunicaciones móviles los que hagan que se dé un paso más allá.

El desarrollo de estándares de la familia IEEE 802.11 (la base de los productos que en el mercado se conocen como Wi-Fi) llegó al terreno que nos ocupa en forma de su versión 802.11p [WAVE2]. Su objetivo fue proporcionar un acceso inalámbrico en ambientes denominados *vehiculares*, para servir como base de posibles aplicaciones orientadas a subsanar gran parte de los problemas anteriormente descritos. Esto incluiría intercambio de información entre vehículos desplazándose a gran velocidad, así como con elementos de la infraestructura de las vías.

Es gracias a estos avances y la apuesta que por ellos hicieron organismos y entidades internacionales, con ejemplos tales como el proyecto DSRC - *Dedicated Short Range Communications* (*Comunicaciones Dedicadas de Corto Rango*)- (véase [DSRC]), reserva de recursos radioeléctricos para este fin [EUC], etc., con los que aparecen las aplicaciones basadas en las bautizadas como *redes vehiculares* y los conocidos como *sistemas de transporte inteligente*.

Con la introducción de estos *sistemas de transporte inteligente*, de los que se hablará en mayor profundidad a continuación, el flujo de tráfico en áreas de alta densidad de población, así como la seguridad en la red viaria, se verán considerablemente mejorados. En este campo, la comunidad investigadora ha ido dedicando más recursos conforme los avances en la tecnología que se requiere (tanto *software* como *hardware*) se van produciendo.

Esta onda expansiva tecnológica, que parte del desarrollo de las comunicaciones inalámbricas y se extiende sobre un muy amplio campo de aplicaciones, será la base de la aparición del ya mencionado concepto de *red vehicular*.

El diseño de los sistemas de comunicación para estas redes será el tema sobre el que se estudiará en esta memoria y para el que se buscarán y analizarán las técnicas de desarrollo. En este sentido, se verá que la simulación va a jugar un papel primordial, pues se trata de una herramienta básica para el trabajo de los equipos de la comunidad desarrolladora. Es gracias a las plataformas de simulación que se podrán probar, estudiar, validar y en último término, implementar, los sistemas de comunicación. Todo ello sin el requerimiento de grandes cantidades económicas, construcción de infraestructura ni, en definitiva, una apuesta *a ciegas* por parte de la industria o de la comunidad investigadora.

La simulación es un punto clave en el devenir de las redes vehiculares, pero, ¿qué debe cumplir un sistema de simulación para ofrecer buenos resultados en este particular mundo de las redes vehiculares, que implica escenarios tan heterogéneos? ¿Cuándo considerarlo realista? ¿Cómo trabajar con él? Todas estas preguntas (y sus respuestas) serán planteadas a lo largo de esta memoria.

1.2 Objetivos

Como ya se ha anunciado, el objetivo fundamental de este proyecto será dotar al lector de los medios y conocimientos para trabajar en el desarrollo de sistemas de comunicación para redes vehiculares. Concretamente, se hará notar la necesidad de los

sistemas de simulación orientados a tal fin, para finalizar ofreciendo al desarrollador, las herramientas de las que precisará.

Se concluirá que el sistema que mejores prestaciones presenta es el marco de simulación VEINS, por lo que el trabajo irá enfocado a exponer las bases sobre las que éste se sustenta y a servir como guía para su uso.

En base a este objetivo principal, se proponen los siguientes objetivos parciales:

- Dar a conocer las redes vehiculares y su amplio abanico de aplicaciones.
- Mostrar la necesidad de las simulaciones de cara al desarrollo de sistemas de comunicación intervehicular.
- Realizar un análisis en profundidad de las características que debe cumplir un sistema de simulación para ser considerado realista.
- Presentar las herramientas de las que se disponen en la actualidad, prestando especial atención a los sistemas de código abierto.
- Desarrollar un estudio exhaustivo del simulador de redes vehiculares VEINS y sus componentes.
- Aportar una completa guía de uso de esta herramienta de simulación, de forma que el desarrollador de sistemas de comunicación intervehicular, disponga de un documento versátil y de gran utilidad para su trabajo y, de esta forma, contribuir con el avance en general de las tecnologías relacionadas con las redes vehiculares.

1.3 Fases de desarrollo y medios empleados

El trabajo aquí expuesto ha sido desarrollado principalmente en cuatro fases diferenciadas, que se explican a continuación.

1. Estudio de las redes vehiculares como base del trabajo posterior: qué son, sus requerimientos tecnológicos, implicaciones, estado de su implementación, etc.

2. Trabajo sobre los sistemas de simulación para redes vehiculares: su necesidad e interés, sus particularidades y estado del arte.

3. Estudio teórico del marco de simulación VEINS. Paradigmas básicos de sus componentes: simulador de red OMNeT++ (y los paquetes requeridos), simulador de tráfico SUMO y módulo de comunicación entre ambos.

4. Trabajo práctico con VEINS: instalación de cada uno de sus componentes, fases de prueba de cara a solventar problemas de incompatibilidad de versiones, así como desarrollo de escenarios reales de simulación y prueba de los mismos.

Cada una de estas cuatro fases principales ha supuesto un pequeño reto en sí misma, pues en la etapas de trabajo práctico surgieron diversos problemas asociados a la alta complejidad de esta herramienta de simulación, así como al hecho de que se requieran múltiples módulos a priori independientes (lo que dio lugar a problemas con las versiones). Por otra parte, las fases de análisis teórico han supuesto la revisión y estudio de una muy importante cantidad de literatura publicada sobre este campo, a partir de la que se han obtenido numerosas conclusiones de enorme interés.

1.4 Medios empleados

En lo que se refiere a los medios empleados (tanto *hardware*, *software*, como recursos de información) para la consecución de este trabajo, pueden resumirse en la siguiente lista. En los siguientes capítulos se dará una explicación detallada de cada uno de los recursos *software* empleados.

- PC Intel Core 2 Duo, 2 GHz y RAM de 4 GB.
- Sistemas Operativos Windows 7 y Debian 6.0.2.1.
- *Software* SUMO versión 0.12.1
- *Software* OMNeT++ versión 4.0 (con los paquetes INET y MiXiM).
- *Software* sistema TraCI.
- Extensa colección de publicaciones de múltiples fuentes, en su mayoría IEEE. En la sección de Referencias se tiene acceso a todas ellas.

1.5 Estructura de la memoria

La memoria se estructura en siete capítulos más un anexo con el código empleado para los ejemplos a los que se hará referencia. Cada capítulo se conforma con el contenido que se muestra a continuación:

Capítulo 1. Introducción y objetivos

Como se ha visto, se presenta el marco general y el problema sobre el que se desea realizar el estudio y se marcan los objetivos que se pretenden cumplir con este trabajo.

Capítulo 2. Redes vehiculares y simulación

Se introduce en profundidad el concepto de red vehicular (también denominada VANET), sus aplicaciones, breve cronología y requisitos tecnológicos. A continuación se presenta la necesidad de los sistemas de simulación, estudiando tanto las herramientas de simulación de red, como las de tráfico rodado. Finalmente, se verá cómo debe ser un simulador para ser considerado realista y qué implica esto en el mundo de las redes vehiculares en concreto.

Capítulo 3. Simuladores de VANETs: Estado del arte

Se verán los tipos de sistemas de simulación que hay presentes en la literatura y se presentarán las principales herramientas de código abierto desarrolladas por la comunidad investigadora hasta el momento.

Capítulo 4. Simulador VEINS

Se realizará un estudio en profundidad del marco de simulación VEINS: sus componentes y paradigmas básicos sobre los que se construyen. Finalmente se hará especial hincapié en la ventaja comparativa que aporta esta herramienta, que es el denominado acoplamiento bidireccional entre el mundo de la simulación de red y el de la simulación de tráfico.

Capítulo 5. Instalación y puesta en marcha

En este capítulo se ofrece al usuario una detallada guía de instalación de cada uno de los componentes de VEINS, así como de algunos auxiliares. Tras seguir los pasos que aquí se describen, el lector será capaz de ejecutar la simulación que desee, tanto si trabaja en un sistema Windows, como si lo hace en Linux.

Capítulo 6. Uso del simulador

Una vez que se dispone del sistema VEINS completo, el usuario estará en disposición de lanzar sus simulaciones. Para ello, el capítulo 6 ofrece un análisis exhaustivo y práctico de las posibilidades que este simulador de redes vehiculares brinda. Se completará mediante el desarrollo de un ejemplo práctico.

Capítulo 7. Conclusiones y trabajo futuro

Para finalizar y a modo de cierre, se hará una recopilación de las conclusiones obtenidas durante el transcurso de la memoria. De igual modo, se presentarán algunas ideas como trabajo futuro que pudiera ser continuación de este.

Capítulo 2

Redes vehiculares y simulación

2.1 Introducción a las redes vehiculares

2.1.1 Definición

A modo de introducción y puesta en contexto, se presentan a continuación los conceptos básicos sobre los que se irán trabajando a lo largo de esta memoria y a partir de los cuales surgen los que serán el pilar de este Proyecto Fin de carrera (los sistemas de simulación). La base de lo que se expondrá no es otra que el mundo de las redes vehiculares

Una red vehicular, de ahora en adelante VANET (de la voz inglesa *Vehicular Ad-hoc Network*), es una clase particular de red móvil (en la literatura denominada MANET: *Mobile Ad-hoc Network*), con la particularidad de estar formada por conjuntos de vehículos que se comunican entre sí mediante equipos de transmisión radio (lo que es conocido como comunicación intervehicular, a partir de ahora IVC, del inglés *Inter-Vehicular Communication*) y, en determinadas ocasiones, también con elementos que forman parte de la infraestructura de las vías de circulación (referidos como *RSU*: *Road*

Side Unit). Se puede ver un ejemplo de los elementos que conforman una de estas redes en la Figura 2.

Las VANETs tienen una serie de características muy determinadas que hacen interesante su estudio dentro del conjunto de las MANETs. Entre ellas, y como una de las principales, está el hecho de que la movilidad que tiene lugar se produce a altas velocidades. Asimismo, se caracterizan por tener un limitado grado de libertad en el movimiento de los nodos, pues obviamente, la topología de la red vial supone un factor clave en este sentido.

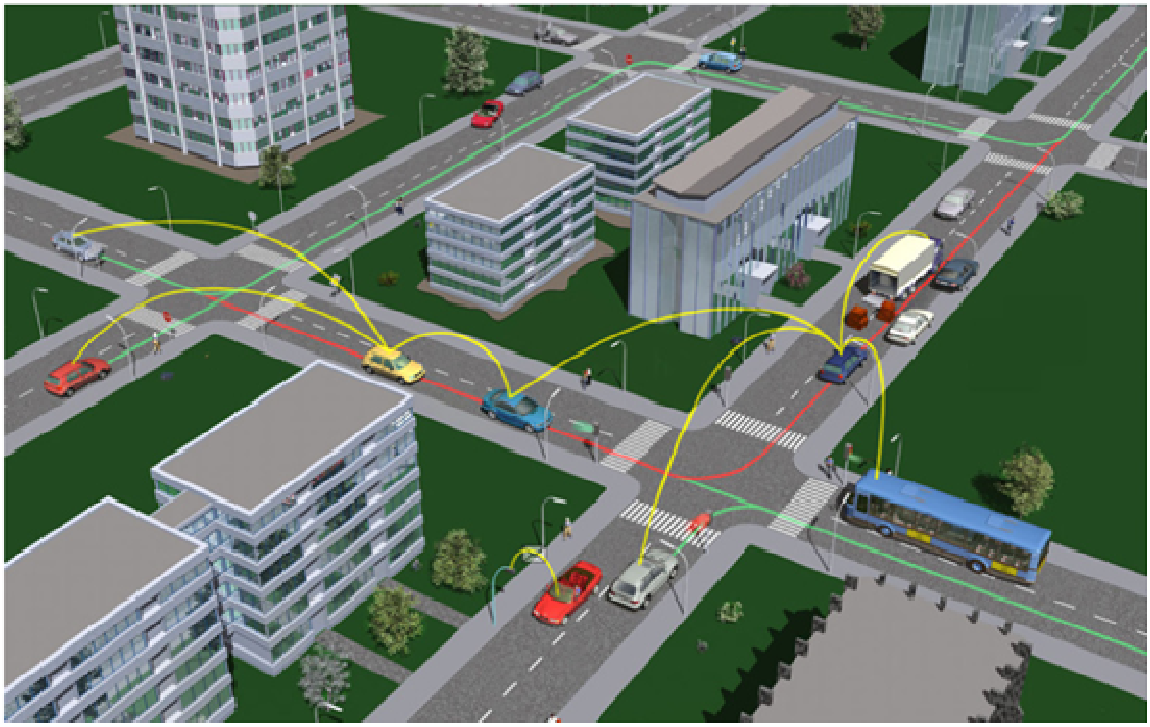


Figura 2. Ejemplo de VANET [Nar08]

Existe un amplio abanico de aplicaciones posibles para este tipo de redes, las cuales se analizarán a continuación, yendo desde la difusión de mensajes de emergencia, hasta la monitorización en tiempo real del tráfico, pasando por aplicaciones relacionadas con la seguridad vial y prevención de colisiones. En todas ellas existirá un intercambio de mensajes entre los vehículos, con el fin de mejorar la capacidad de respuesta del conductor y por consiguiente, la seguridad en el caso de que haya algún incidente en la carretera. No obstante, también puede facilitar el desarrollo de aplicaciones que faciliten la comodidad durante el trayecto a recorrer, como se verá a continuación.

Este conjunto de funcionalidades que las VANETs son capaces de proporcionar, hacen de las mismas un componente fundamental de los denominados Sistemas Inteligentes de Transporte (ITS, del inglés *Inteligent Transportation Systems*) [ITS]. Este concepto surge con fuerza como consecuencia de los problemas derivados del tráfico y del importante crecimiento de las tecnologías de la información. El Departamento de Transporte de los EE.UU. define a los ITS como sistemas que recogen, almacenan, procesan y distribuyen información relacionada con el movimiento de personas y bienes. De forma concisa y muy explicativa, se refiere a ellos con la siguiente descripción [ITSA]: "*se trata de gente usando tecnología en transportes para salvar vidas, tiempo y dinero*".

Las particulares características ya mencionadas de las VANETs, tienen como consecuencia que normalmente los protocolos habituales utilizados para otras aplicaciones de redes móviles, resulten ineficientes. Es por este motivo por el que en los últimos años, los investigadores han trabajado y siguen trabajando en el desarrollo de nuevos protocolos de comunicación [MLP09]. Será concretamente a este campo hacia el que irán enfocados los siguientes capítulos, más específicamente a los sistemas de simulación utilizados para el estudio y prueba de nuevas soluciones tecnológicas.

2.1.2 Aplicaciones

En esta sección se plantea una lista de las principales aplicaciones de las VANETs, así como los requisitos que deben cumplir. Se pretende facilitar una visión lo más completa posible de los servicios que estas redes son capaces de prestar, para que así el lector tome conciencia del impacto que pueden llegar a tener y, por lo tanto, de su elevada importancia.

Siguiendo la clasificación que se presenta en [Kar11], podemos distinguir las siguientes aplicaciones y casos de uso principales:

1. **Seguridad vial**

Se trata de aquellas aplicaciones que tienen como finalidad disminuir los accidentes de tráfico y, por tanto, la pérdida de vidas humanas [DOT].

Un importante porcentaje de los accidentes que tienen lugar en todo el mundo está asociado a las intersecciones de vías y colisiones frontales y laterales de vehículos [Kar11]. Este tipo de aplicaciones proveen información y asistencia al conductor, de cara a evitar dichos choques. Si esto, además, se complementa mediante el intercambio de información (posición del vehículo, posición de las intersecciones, velocidad, etc.) entre

vehículos y *RSUs*, podría incluso utilizarse para predecir colisiones. Es más, esta información permite localizar puntos peligrosos en las carreteras, tales como lugares resbaladizos o con baches.

Se recogen, a continuación, algunos de los principales casos de uso existentes:

- ***Advertencias en intersecciones***

En este caso, se detecta el riesgo de choque lateral para vehículos que se aproximan a una intersección. Los vehículos ya presentes en la intersección y/o los *RSUs*, detectan este peligro e informan a quienes se están aproximando.

- ***Asistencia para el cambio de carril***

Se trata de reducir el riesgo de colisión lateral, debido a los denominados puntos ciegos, cuando se realiza un cambio de carril. Éste es un problema especialmente preocupante para el caso de camiones, como se puede observar en la Figura 3, en la que se representa una situación potencialmente peligrosa (las porciones sombreadas corresponden a las áreas o puntos ciegos).

- ***Advertencias durante adelantamientos***

El objetivo es evitar choques laterales en el caso de adelantamientos. La situación es la siguiente: un vehículo (v_A) trata de adelantar a otro (v_C), mientras que un tercero (v_B) ya ha comenzado la maniobra de adelantamiento. La colisión entre v_A y v_B puede ser prevenida, por medio de un mensaje de v_C a v_A pidiendo que detenga el proceso de adelantamiento.

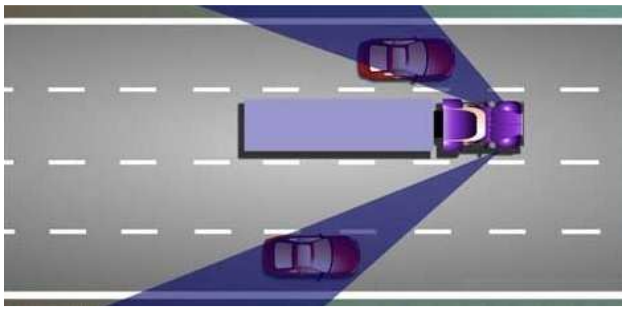


Figura 3. Caso de uso: asistencia para el cambio de carril

- ***Advertencia de posible colisión frontal***

El riesgo de una colisión frontal es reducido mediante el envío de avisos que informen de presencia, a los vehículos que circulan en dirección contraria.

- ***Advertencia de posible colisión trasera***

El conductor es informado de un riesgo de colisión trasera con el vehículo que se encuentra delante, posiblemente causado por una frenada brusca, falta de visibilidad en curvas, cambios de rasante, etc.

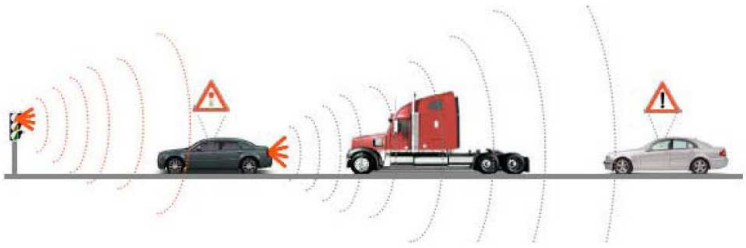


Figura 4. Ejemplo de aplicación correspondiente al Proyecto CVIS de la Comisión Europea [CVIS]

- ***Advertencia conjunta de colisión con vehículo precedente***

Se trata de una situación similar a la anterior, pero en este caso, el riesgo de colisión es detectado mediante la cooperación de ambos vehículos implicados. El sistema deberá asistir a los conductores con el fin de evitar el choque.

- ***Preaviso de choque***

En caso de hacer uso de este mecanismo, significaría que el choque es inevitable y por tanto, tendrá lugar. Vehículos y *RSUs* comparten información periódicamente con el fin de pronosticar colisiones. La información compartida incluye datos sobre la localización y el tamaño del vehículo con el propósito de capacitar y optimizar el equipamiento del vehículo para disminuir los efectos de la colisión. Este equipamiento puede estar compuesto por air bags, cinturón de seguridad con tensores y parachoques extensibles, etc.

- ***Asistencia en confluencias de vías***

Los vehículos que se encuentran en una confluencia de vías negocian y cooperan ente ellos y con *RSUs* para dar cuenta de su maniobra y evitar colisiones.

- ***Aviso de frenado de emergencia***

Vehículos que tienen que frenar bruscamente informan sobre esta situación al resto de vehículos mediante la cooperación de otros vehículos y/o a través de *RSUs*.

- ***Aviso de circulación en sentido contrario***

Un vehículo que detecta que está circulando en sentido contrario, por ejemplo, dirección prohibida, señala esta circunstancia a otros vehículos y a los *RSUs*.

- ***Alerta de vehículo estacionado***

En uso de este mecanismo, cualquier vehículo que esté parado, debido a un accidente, avería o cualquier otra razón, informa sobre su situación a los demás vehículos y a los *RSUs*.

- ***Aviso de las condiciones del tráfico***

Cualquier vehículo que detecte una evolución/cambio rápido de la fluidez del tráfico informa sobre su situación a los demás vehículos y a los *RSUs*.

- ***Violación de señal de tráfico***

Uno o más *RSUs* detectan violaciones de señales de tráfico. Esta circunstancia es retransmitida por los *RSUs* a todos los vehículos de los alrededores, de cara a advertirlos de un posible escenario peligroso.

- ***Notificación de punto peligroso***

Cualquier vehículo o *RSU* informa a otros vehículos sobre la existencia de puntos peligrosos, tales como obstáculos en la carretera, obras o calzadas deslizantes (Figura 5*)*.

- ***Aviso de pérdida de control***

En este caso, se trata de habilitar al conductor de un vehículo para informar sobre una posible pérdida de control a los vehículos que le rodean. Una vez recibida esta información, el resto de vehículos determinan la relevancia del evento y proceden al aviso a los conductores si fuera necesario.

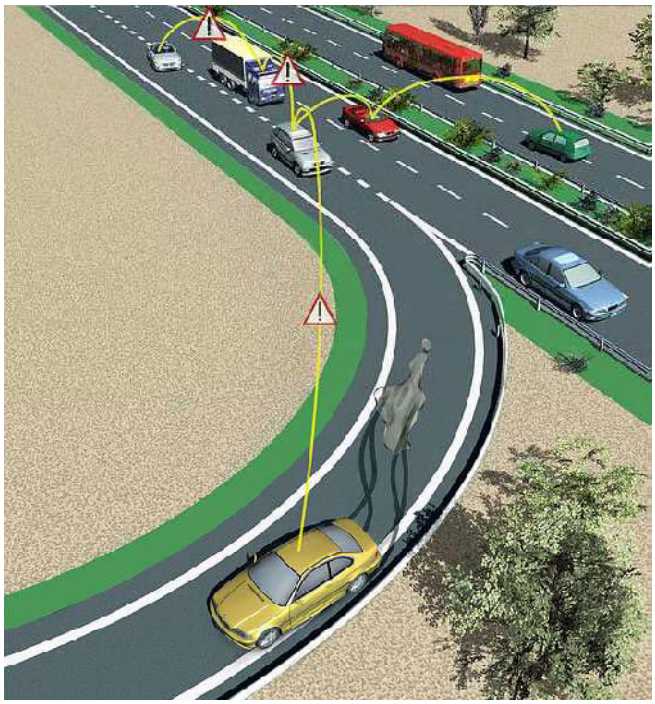


Figura 5. Aviso de elemento o punto peligroso en la carretera [CLIFF]

2. Gestión y mejora de eficiencia del tráfico y la conducción

Estas aplicaciones están enfocadas a mejorar el flujo de tráfico, la coordinación de éste y la asistencia en carretera. De igual modo pretende proveer información local actualizada, mapas y mensajes relevantes en tiempo y espacio. *Gestión de la velocidad* y *navegación cooperativa* son dos de los grupos en los que típicamente se clasifican este tipo de sistemas.

- ***Gestión de la velocidad***

Tienen por objetivo asistir al conductor para controlar la velocidad de su vehículo de cara a lograr una conducción suave y para evitar paradas innecesarias. Un regulador de velocidad o avisos sobre la velocidad óptima mediante una luz verde son dos ejemplos.

- ***Navegación cooperativa***

Son usados para incrementar la eficiencia del tráfico mediante la cooperación de los navegadores de los vehículos entre ellos o entre éstos y los *RSUs*. Algunos ejemplos de este tipo son: información del tráfico, itinerario provisional recomendado, control de rutas adaptadas en cooperación y sectorización.

3. Servicio público

En este caso, el objetivo es facilitar la labor de los cuerpos de seguridad y emergencias públicos. Se pueden diferenciar, por consiguiente, dos grupos fundamentales.

- ***Aviso de vehículo de emergencias***

Un vehículo de emergencias en activo, por ejemplo, ambulancias o coches de policía, señalan al resto de vehículos en la manzana que deben dejar libres los carriles. Esta información puede ser retransmitida en el mismo entorno por otros vehículos y los *RSUs* disponibles en la zona.

- ***Apoyo a las autoridades***

Como herramienta para el control policial de los vehículos que circulan por la carretera, o incluso de la licencia de los conductores. Esto puede suponer la fácil localización de vehículos robados o la inspección de vehículos de seguridad.

4. Sistemas de información/entretenimiento

- ***Servicios locales cooperativos***

Este tipo de sistemas está enfocado a la información/entretenimiento que puede ser obtenida de los servicios locales, como podrían ser notificaciones sobre los puntos de interés, comercio electrónico local y descargas periodísticas.

- ***Servicio de Internet***

Centrados en información que puede ser obtenida de Internet. Ejemplos típicos son: *Servicios Comunitarios*, que incluyen seguros y servicios de financiación, gestión de flota y gestión de parkings, y *ITS station life cycle*, centrado en software y actualización del mismo.

A continuación se puede ver la Tabla 1, a modo de resumen, donde se muestra de forma concisa y gráfica, la gran variedad de campos de aplicación que se ha presentado, así como un ejemplo de cada uno de ellos.

	Propósito	Ejemplo
Seguridad Vial	Carreteras peligrosas	Advertencia de intersecciones
	Condiciones anormales de tráfico	Advertencia de zona de obras
	Peligro de colisión	Advertencia de ángulo muerto; cambio de carril; peatones que cruzan
	Choque inminente	Frenada pre-accidente
	Aviso de incidente	Alerta tras colisión ya ocurrida. Aviso SOS
Servicio Público	Respuesta de emergencia	Alerta vehículo de emergencias, señal de anticipación
	Apoyo a las autoridades	Matrícula o licencia de conducir electrónica
Mejora de la Conducción	Mejora de la conducción	Asistente de carretera o giro
	Eficiencia del tráfico	Notificación accidentes; guía de ruta y navegación; aparcamientos
Negocios y Entretenimiento	Mantenimiento de vehículos	Diagnósticos remotos; avisos de seguridad
	Servicios móviles	Internet; notificación de puntos de interés
	Soluciones empresariales	Gestión de flotas; alquiler; seguimiento de la carga

Tabla 1. Campos de aplicación y ejemplos de las VANETs

2.1.3 Cronología

La idea de desarrollar comunicaciones inalámbricas entre vehículos viene siendo promovida por los investigadores desde la década de los 80. No obstante, ha sido en los últimos años cuando hemos podido presenciar el mayor auge en la investigación sobre este campo y, por consiguiente, en su desarrollo.

Diversos factores han sido clave para ello, entre otros, la amplia adopción de las tecnologías IEEE 802.11 (como se ha mencionado en el anterior capítulo); la tendencia de la industria de la automoción hacia las tecnologías de la información para afrontar los problemas de la seguridad vial, medioambientales y de confort de los pasajeros; así como el compromiso de gobiernos nacionales y regionales de cara a gestionar la asignación del espectro radioeléctrico para comunicaciones vehiculares inalámbricas.

Desde finales de los años 90, existen a nuestra disposición equipos de bajo coste para posicionamiento global (*GPS*), así como receptores/transmisores de red de área local inalámbrica (WLAN). Esto también ha supuesto un factor fundamental para que multitud de proyectos se hayan llevado a cabo. En estos proyectos de cooperación para explorar el potencial de las VANETs, se han visto involucrados distintos actores: industria del automóvil, entidades operadoras de las vías de circulación, agencias de control de peajes y otros proveedores de servicios. Todo ello, normalmente financiado por gobiernos nacionales, los cuales también han contribuido facilitando licencias para el reparto del espectro, generalmente en la banda de 5,8/5,9 GHz (y la de 700 MHz en Japón).

En la siguiente Figura 6, podemos ver los principales hitos alrededor del mundo en el desarrollo de las redes vehiculares. Debido a que el número de proyectos que han surgido en los últimos años es realmente importante, se muestran a continuación únicamente las principales iniciativas desde los comienzos hasta la consolidación del concepto de VANET.

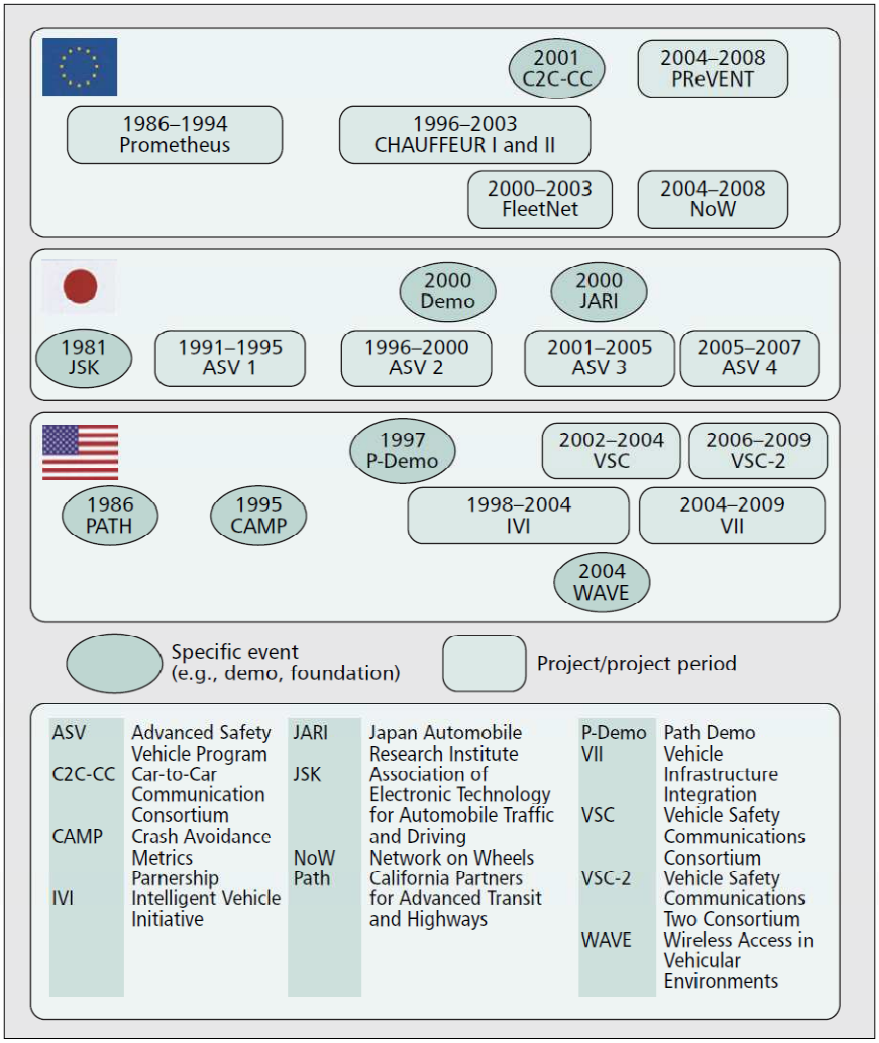


Figura 6. Actividades pioneras y primeros hitos en la investigación en VANETs [Han08]

2.1.4 Requerimientos tecnológicos

Los requerimientos tecnológicos a los que se harán referencia son aquellos derivados del estudio de las necesidades de las aplicaciones de red que se han presentado. Estos requerimientos pueden clasificarse en los siguientes grupos [ETSI]:

a. Requerimientos estratégicos:
   Este tipo de requerimientos están relacionados con el nivel de desarrollo de las redes móviles destinadas al campo de las VANETs.

b. Requerimientos económicos:
   Estrechamente relacionados con los anteriores, en lo que respecta a su enfoque financiero, de negocio, valor percibido por el cliente, coste y posible retorno de las inversiones.

c. Capacidad de los sistemas:
   En lo que se refiere a los requerimientos técnicos propiamente dichos, podríamos distinguir los siguientes:

   Radiocomunicaciones
   Entre las que cabe mencionar el rango de trabajo para comunicaciones radio de único salto, canales de RF disponibles, ancho de banda (tasa binaria), nivel de compensación para la propagación de la señal debido a los obstáculos, etc.

   Parámetros telemáticos
   Tales como la modalidad de comunicación a utilizar: *unicast*, *multicast*, *broadcast*, *geocast* (*broadcast* únicamente en un área específica); la gestión de la congestión; gestión de la priorización de mensajes; compatibilidad con el direccionamiento IPv4 e IPv6, etc.

   Posicionamiento del vehículo
   Sistemas como GNSS (*Global Navigation Satellite System*): GPS (*Global Positioning System*), o combinaciones de GNSS con información provista por mapas geográficos locales.

   Seguridad en comunicaciones
   Respeto a la privacidad y anonimato, integridad y confidencialidad, resistencia a ataques externos, autenticidad de los datos recibidos, integridad de los sistemas, etc.

d. Rendimiento de los sistemas:
   Los requerimientos en este ámbito son relacionados, principalmente, con el tiempo de latencia en las comunicaciones, frecuencia de actualización, precisión del posicionamiento, fiabilidad del sistema, cobertura de las comunicaciones radio, tasa de error de bit y rendimiento de los sistemas de seguridad (verificación de los mensajes mediante certificados, por ejemplo).
e. Requerimientos organizativos:
   En este sentido, se debe tener en cuenta el papel que juegan las organizaciones y sistemas de estandarización. Como requerimientos habría que destacar el desarrollo de un directorio para direccionamiento y nomenclatura común y consistente, esquemas de asignación de direcciones IPv4 ó IPv6, una organización de soporte para los requerimientos de seguridad, etc.

f. Requerimientos legales:
   En lo que respecta a responsabilidades legales, soporte y respeto a la privacidad de los usuarios.

Podemos ver algunos de los requerimientos específicos para una serie de aplicaciones de interés en la Tabla 2.

Caso de Uso	Modo de Comunicación	Frecuencia mínima de transmisión	Latencia crítica
Advertencia de colisión en intersección	*Broadcasting* periódico de mensaje	10 Hz	< 100 ms
Asistencia al cambio de carril	Cooperación entre vehículos	10 Hz	< 100 ms
Aviso de adelantamiento	*Broadcast* de “estado adelantando”	10 Hz	< 100 ms
Aviso de posible colisión frontal	*Broadcasting* de mensaje	10 Hz	< 100 ms
Aviso cooperativo de posible colisión frontal	Cooperación entre vehículos asociada a *unicast*	10 Hz	< 100 ms
Aviso de vehículo de emergencias	*Broadcasting* permanente y periódico de mensaje	10 Hz	< 100 ms
Aviso de riesgo de colisión	Mensaje periódico limitado en el tiempo	10 Hz	< 100 ms

Tabla 2. Requerimientos para aplicaciones asociadas a la seguridad vial [Kar11]

2.2 Simulaciones

2.2.1 Necesidad de las simulaciones

El creciente interés hacia las posibles aplicaciones de las tecnologías inalámbricas en el terreno vehicular ha llevado a la relativamente reciente creación de consorcios ([VII], [C2C]) y organismos de estandarización (algunos bajo la tutela del IEEE [WAVE]) con el fin de desarrollar tecnologías y protocolos para la transmisión de datos entre vehículos y entre vehículos e infraestructura de las vías de circulación.

Como ha quedado patente, las VANETs suponen un caso particularmente complejo de MANETs, caracterizadas por muy altas velocidades y un muy limitado grado de libertad en los patrones de movimiento de los nodos. Estas particularidades normalmente hacen de los protocolos tradicionales de comunicación telemática una solución ineficiente o incluso en ocasiones inservible para las VANETs.

Es evidente la fuerte interacción entre el protocolo de red y la movilidad de los vehículos. En la versión estándar de un sistema de comunicaciones móviles, el tráfico de datos se ve alterado por la movilidad, pero en el caso de las VANETs esta interrelación aún va más allá: la movilidad también se ve alterada por el tráfico de datos. Éste es el fin último de este tipo de redes, pues los datos intercambiados entre los nodos pueden (o deben) dar lugar a que su movimiento cambie (se altere su velocidad, el recorrido, el patrón de movimiento, etc.).

Considerando el gran impacto que el desarrollo de las tecnologías VANET puede ocasionar en el mercado de la automoción, es fácilmente comprensible el enorme esfuerzo que se está realizando actualmente en el desarrollo de protocolos de comunicaciones y modelos de movilidad específicos para las redes vehiculares.

Pese a que es fundamental probar y evaluar los protocolos implementados en escenarios reales, existen dificultades logísticas obvias, problemas económicos y limitaciones tecnológicas que hacen de las simulaciones el medio más comúnmente elegido para validar los protocolos de comunicaciones que se desarrollan para las VANETs.

Un aspecto crítico en el estudio mediante simulación es la necesidad de un modelo de movilidad que refleje el comportamiento real del tráfico de vehículos. De hecho, se requerirá de modelos de movilidad que sean dinámicamente reconfigurables, con el fin de reflejar los efectos de un protocolo de comunicación en particular sobre el tráfico viario.

La comunidad investigadora, por tanto, comenzó a trabajar en el desarrollo de modelos de movilidad específicos para los movimientos vehiculares. Tras varios años

muy productivos en este sentido, tenemos a nuestra disposición una importante variedad de modelos, desde el más trivial al más realista, o desde los disponibles gratuitamente a los modelos comerciales. Desgraciadamente, este desarrollo se ha producido de forma descoordinada [Har09], pues cada grupo de investigación básicamente se ha dedicado a desarrollar un modelo que satisficiera sus propias necesidades específicas.

Por otra parte, debemos prestar especial atención, igualmente, a los simuladores de red, pues un sistema de simulación para VANETs está compuesto por ambos bloques, como se desarrolla a continuación.

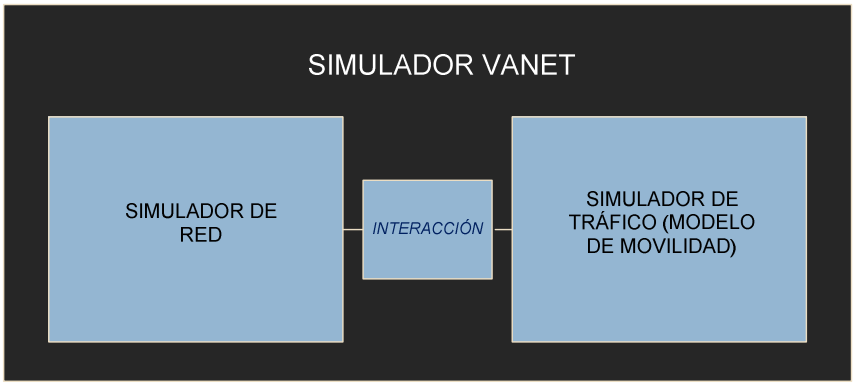


Figura 7. Estructura básica de un sistema de simulación para VANETs

2.2.2 Simuladores de red

En el contexto del estudio de una red de transmisión de datos, construir bancos de pruebas y laboratorios es caro, reconfigurarlos, reutilizarlos o compartirlos es difícil, pues son relativamente poco flexibles. Añadido a esto, está el hecho de que reproducir ciertos fenómenos presentes en las redes, tales como interferencias en comunicaciones radio, resulta complicado.

Los simuladores de red multiprotocolo suponen una gran oportunidad para realizar una experimentación eficiente. Entre sus beneficios, podemos encontrar [Lee00]:

- Validación del comportamiento de los protocolos existentes.
- Infraestructura para desarrollar nuevos protocolos.
- Posibilidad de comparación de los resultados para diferentes protocolos.
- Oportunidad de estudiar protocolos con interacción a gran escala en un entorno controlado.

La mayoría de los simuladores de red utilizan la simulación de eventos discretos, en la que los eventos son almacenados en una lista de espera, y se procesan en orden.

Los simuladores de red, como su nombre indica, son utilizados por los investigadores, desarrolladores e ingenieros para diseñar distintos tipos de redes, simular y analizar el efecto de diversos parámetros y protocolos sobre el rendimiento de la red. Un simulador de red típicamente abarca una amplia gama de tecnologías de red y puede ayudar a los usuarios a construir redes complejas a partir de bloques básicos, tales como una variedad de tipos de nodos y enlaces. Con la ayuda de simuladores, se pueden diseñar redes jerárquicas que utilizan diversos tipos de nodos: ordenadores, *hubs*, *routers*, *switches*, enlaces, dispositivos móviles, etc.

Con ayuda de un simulador, típicamente, podremos estudiar múltiples tipos de tecnologías de red de área amplia (WAN), tecnologías como TCP, ATM, IP, etc., así como de red de área local (LAN), como Ethernet, Token rings, etc. Asimismo, el usuario puede probar y analizar el resultado de diversos estándares, aparte del estudio de las prestaciones de un nuevo protocolo que haya sido desarrollado.

Hay una amplia variedad de simuladores de red, que van desde el muy simple al muy complejo, en el capítulo siguiente se presentará un análisis taxonómico de los mismos. Como mínimo, un simulador de red debe permitir al usuario representar una topología de red, especificando los nodos de la misma, los enlaces entre los nodos y el tráfico entre éstos. Los sistemas más complicados pueden permitir que el usuario especifique todo lo relacionado con los protocolos que manejan el tráfico de la red. Cabe destacar que las aplicaciones gráficas permiten a los usuarios visualizar fácilmente el funcionamiento de su entorno simulado. Por otra parte, aplicaciones basadas en texto puede proporcionar una interfaz menos intuitiva, pero pueden permitir formas más avanzadas de personalización.

2.2.3 Modelos de movilidad: Simuladores de tráfico

Se desarrollará a continuación un estudio comparativo de los distintos modelos de movilidad disponibles expresamente para las redes vehiculares.

En general, el desarrollo de los modelos de movilidad vehicular puede clasificarse en cuatro clases diferentes [Har09]: *Modelos Sintéticos,* donde englobaremos a todos las soluciones basadas en modelos matemáticos; *Modelos basados en datos,* los cuales extraen los patrones de movilidad de datos reales obtenidos mediante estudios/sondeos; *Modelos basados en trazas*, que generan patrones de movimiento basándose en trazas

reales y finalmente, *Modelos basados en simuladores de tráfico*, para los que las trazas de movimiento vehicular se obtienen de un simulador de tráfico detallado.

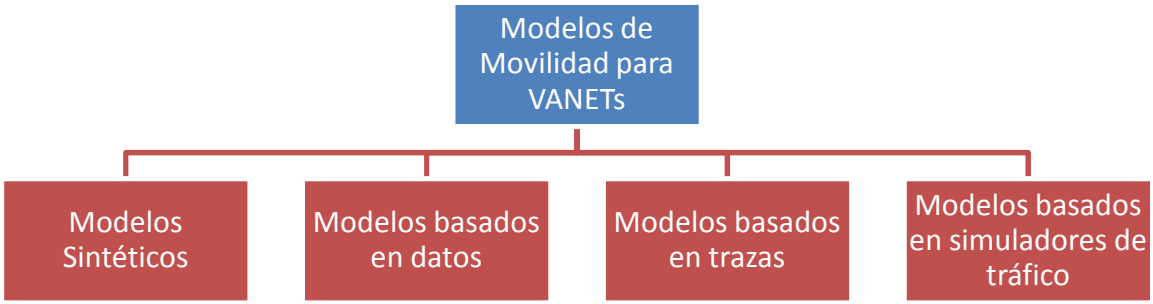


Figura 8. Clasificación de los modelos de movilidad vehiculares

Modelos sintéticos

La primera y más conocida clase es la que engloba a todos los modelos sintéticos. En los últimos años se han llevado a cabo importante estudios con el fin de desarrollar modelos matemáticos que reflejen un efecto físico real.

Dentro de este extenso grupo, podemos separar a los modelos en cinco clases [Fio08] (ver Figura 9): *modelos estocásticos* (movimientos puramente aleatorios), *de flujo de tráfico* (basados en fenómenos de hidrodinámica), *de coche seguidor* (el comportamiento de cada conductor es modelado de acuerdo con el movimiento de los vehículos precedentes), *de colas* (los cuales modelan las carreteras como colas FIFO y los coches como clientes) y modelos *de comportamiento* (cada movimiento está determinado por normas de comportamiento, tales como influencias sociales).

La validación de un modelo matemático es un paso muy importante para garantizar su realismo. Una solución para esto es conseguir trazas de movimientos reales, obtenidas durante campañas de mediciones, y comparar los patrones observados con los desarrollados por el modelo.

Una limitación fundamental de la mayoría de los modelos sintéticos radica en la complejidad de modelar comportamientos humanos al detalle. Los conductores no son en absoluto máquinas y no pueden programarse para seguir un comportamiento específico en todos los casos. Por el contrario, su respuesta a los estímulos y perturbaciones puntuales puede tener un efecto global en el modelado del tráfico. Por lo tanto, un modelo de movilidad realista debe tener en cuenta lo que podemos llamar una “teoría del

comportamiento". Algunos autores han trabajado sobre este campo, desarrollando modelos sintéticos basados en teoría de redes sociales, demostrando ser buenas aproximaciones a los patrones de movimiento humanos [Mus06].

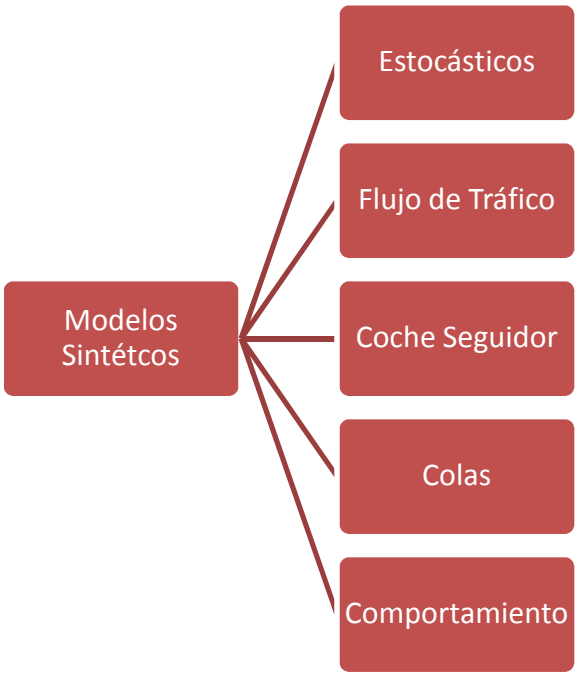


Figura 9. Clasificación de los modelos sintéticos [Fio08]

Para finalizar, la idea general tras los modelos de movilidad sintéticos es tratar de entender un movimiento en particular, desarrollar entonces un modelo matemático, para finalmente reproducir el movimiento de forma "artificial". Cabe destacar, no obstante, que en ocasiones determinados movimientos, o mejor dicho las interacciones entre ellos, hacen del modelo matemático una herramienta demasiado compleja o incluso no realizable.

Modelos basados en datos

La recopilación de datos reales es una fuente importantísima de información sobre la movilidad macroscópica. La fuente de datos a gran escala principales en este campo es el Departamento de trabajo de los Estados Unidos, el cual ha aportado extensas estadísticas sobre el comportamiento de los trabajadores norteamericanos, relacionadas con el tiempo dedicado al transporte al trabajo, distancia recorrida, etc. Incluyendo este tipo de

estadísticas en un modelo de movilidad, se podría generar un modelo genérico capaz de reproducir el comportamiento pseudo-aleatorio o determinista que se podría observar en un tráfico urbano real. Estos son los denominados *modelos basados en datos*.

Existen múltiples modelos basados en datos a nuestra disposición, entre ellos cabría mencionar el modelo UDel [UDel] y el desarrollado por el ETH de Zurich [RVT]. No obstante, si se necesita un modelo más realista y detallado, se deberá acudir al modelo sintético, calibrado usando datos reales.

Modelos basados en trazas

Debido a la dificultad de modelar los movimientos vehiculares, solo algunos modelos sintéticos son capaces de lograr una solución cercana a los patrones de movilidad reales. Sin embargo, se puede enfrentar el problema de una forma diferente: en lugar de desarrollar modelos matemáticos complejos para después calibrarlos usando trazas de movilidad reales, se pueden extraer patrones de movilidad genéricos a partir de trazas reales.

Esta solución ha ganado popularidad en los últimos años. Entre otros, cabe destacar el proyecto *Reality Mining* del MIT [RML].

La parte más compleja de esta variante radica en el proceso de extrapolación de los patrones no observados directamente en las trazas. Mediante el uso de modelos matemáticos, es posible predecir hasta cierto punto los patrones no presentes en las trazas. La limitación normalmente está relacionada con el tipo de campaña de medición que se utilice. Por ejemplo, si las trazas de movimiento han sido obtenidas para autobuses, no se podría obtener un modelo extrapolado para vehículos personales.

Modelos basados en simuladores de tráfico

Refinando los modelos sintéticos y llevando a cabo un proceso intenso de de validación basado en trazas reales y datos sobre comportamiento, algunos equipos de investigación han conseguido desarrollar auténticos simuladores de tráfico realistas. Más adelante, en el capítulo siguiente, se expondrá un análisis en detalle de los simuladores que podemos encontrar, sus desarrolladores y sus prestaciones.

Este último grupo supone la evolución de los anteriores y uno de los núcleos principales y motivaciones del presente estudio. Es por ello que se dedicará un capítulo

completo a su análisis; y los consecutivos, a la integración con un simulador de red, fijándonos concretamente en el sistema VEINS.

2.2.4 Simulaciones de VANETs realistas

A continuación se prestará especial atención a las características que debe cumplir un simulador para poder ser utilizado en el ámbito de las VANETs con la garantía de obtener unos resultados realistas.

En primer lugar nos centraremos en la pieza más delicada para este tipo de sistemas de simulación: el modelo de movilidad. Tras ello, se hará una reflexión acerca del problema que surge al requerir de la intercomunicación entre el simulador de red y el de tráfico. Este último punto, como se verá, supondrá la principal aportación del sistema VEINS, que más adelante se desarrollará en detalle.

Modelos de movilidad realistas

Como se ha explicado, el diseño de protocolos en el área de la investigación de las VANETs depende fuertemente de las técnicas de simulación empleadas. En escenarios MANET, como es el caso de las VANETs, para lograr resultados realistas es de vital importancia tener en cuenta, además de los protocolos de red utilizados, el modelado de la movilidad de los nodos.

La movilidad de los nodos tiene una fuerte influencia en el resultado de la simulación, por lo que en el proceso de evaluación de un protocolo, la elección del modelo de movilidad es tan importante como lo exacto que sea el propio modelado del protocolo bajo test. Como ya se ha visto, se cuenta con una importante comunidad científica que trabaja con el modelado del movimiento de los vehículos, así como del tráfico de red.

En concreto, un modelo de movilidad que pretenda generar patrones de movimiento vehicular realistas, debe incluir los siguientes bloques fundamentales [Har09]:

- Mapas topológicos realistas y precisos:
  La topología de las vías debe contemplar diferentes densidades de intersecciones, contener múltiples carriles, distintas categorías de calle y sus correspondientes limitaciones de velocidad.

- Obstáculos:
  El concepto obstáculo debe entenderse en sentido amplio, es decir, tanto en lo que respecta a limitaciones para la movilidad, como en lo que se refiere a complicaciones para una comunicación inalámbrica.

- Puntos de atracción/repulsión:
  Los puntos de inicio y final de un recorrido no son aleatorios. La mayoría de las veces, los conductores se mueven a destinos finales similares (puntos de atracción), por ejemplo zonas de oficinas, o desde puntos iniciales (puntos de repulsión) comunes, como pueden ser zonas residenciales. Este hecho típicamente da lugar a que se produzcan cuellos de botella.

- Características de los vehículos:
  Cada tipo de vehículo tiene sus propias características, las cuales tienen un impacto importante en múltiples parámetros del tráfico. Entre ellas podemos encontrar vías en las que un tipo de automóvil no está permitido (por ejemplo, zonas urbanas en las que no pueden circular camiones), o simplemente se debe tener en cuenta que las capacidades de aceleración y deceleración de vehículos pesados es muy diferente a la de los turismos.

- Patrones de viaje:
  La selección del viaje a realizar dependerá de los intereses que mueven a cada conductor. Se denomina viaje al conjunto de puntos de atracción y repulsión en una zona urbana.

- Ruta:
  Se denomina ruta al conjunto de vías que es preciso atravesar para realizar un determinado viaje. La ruta no se escoge de forma aleatoria y, además, puede variar en función de una serie de parámetros, tales como la hora del día, la congestión actual, limitaciones de velocidad, o incluso por los hábitos personales del conductor.

- Aceleración/deceleración progresiva:
  Los vehículos en la realidad no frenan y aceleran de forma abrupta. Esto debe ser considerado por el modelo.

- Patrones de conducción humana:
  Los conductores interaccionan con el entorno, no sólo en lo que respecta a los obstáculos estáticos, sino también con los dinámicos (tales como otros

vehículos, o los peatones). El modelo debe tener en cuenta este hecho y gestionar la interacción entre los distintos agentes (adelantamientos, atascos, etc.).

- Gestión de intersecciones:
  El proceso de control de las intersecciones es fundamental en un modelo. Tendrá efectos en la movilidad de los vehículos, actuando como una restricción más a la misma.

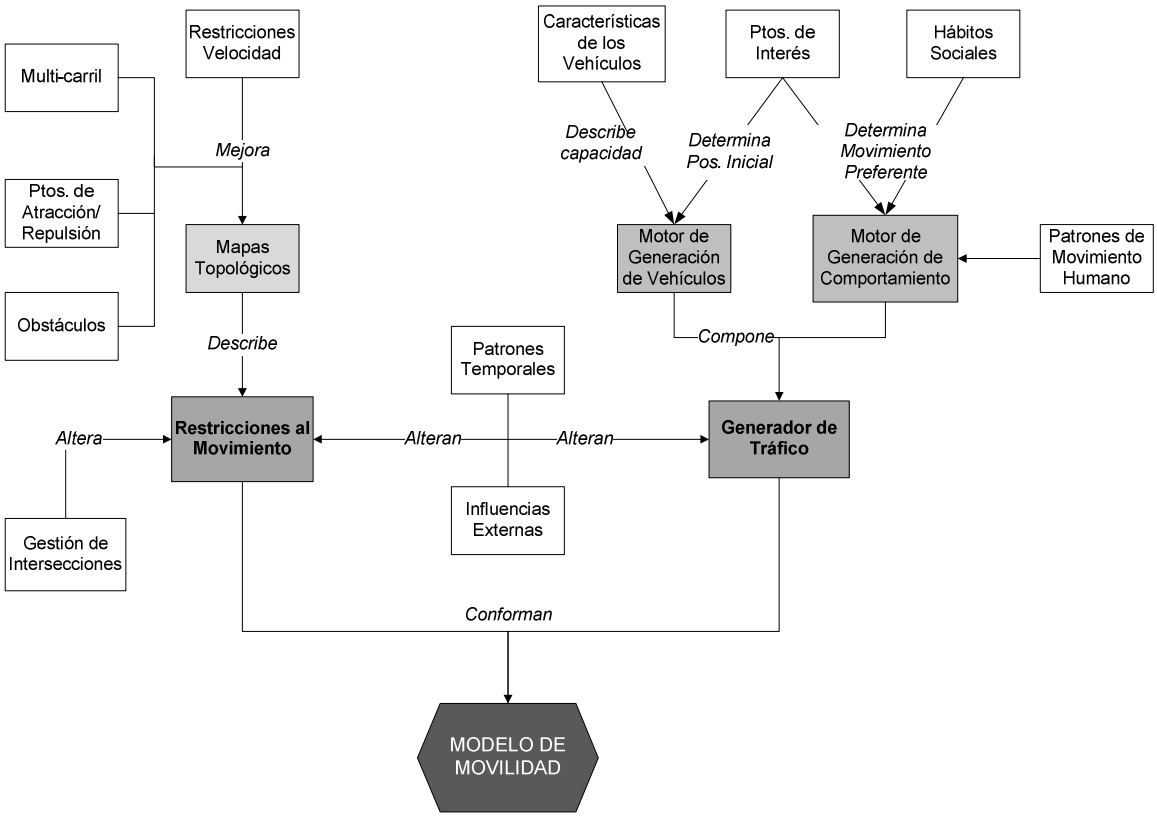


Figura 10. Mapa conceptual para la generación de un modelo de movilidad realista [Har09]

- Patrones temporales:
  La intensidad de tráfico no es la misma en todas las horas del día. En horas punta o eventos especiales, siempre se observa un repunte en el flujo de desplazamientos.

- Influencias externas:
  Esta categoría modela el impacto de los accidentes, trabajos temporales en una vía, o incluso datos en tiempo real del estado del tráfico.

En la Figura 10 anterior podemos ver de forma esquemática todos los bloques anteriormente desarrollados, así como la interacción entre ellos, de cara a obtener un simulador de tráfico realista.

Simuladores de tráfico y red

Veremos en las siguientes secciones de esta memoria que podemos lograr simulaciones más realistas y precisas si los resultados de estas dos ciencias, a priori independientes, se acoplan entre sí, haciendo que la simulación de tráfico esté integrada en la simulación de red.

En la literatura podemos encontrar referencias a este problema describiéndolo, de forma muy gráfica, como "*el mudo hablándole al sordo*". Efectivamente, en el terreno de las VANETs, para lograr una simulación realmente útil (realista), es preciso que el modelo de movilidad (simulador de tráfico) se ponga a disposición del simulador de red.

Este asunto de compatibilidad entre ambos entes es el punto clave a tratar, ya que ha supuesto un importante interrogante y foco de trabajo de múltiples equipos de investigación. De ahí el símil al que se hacía referencia.

En el siguiente capítulo, en el que se tratará el estado del arte de los simuladores, se estudiará en detalle cada una de las soluciones propuestas, viendo las diferentes formas de afrontar el problema y, por consiguiente, las distintas plataformas de simulación que tenemos a nuestro alcance.

El curso de esta memoria nos llevará a la solución propuesta por el grupo de trabajo de Christoph Sommer en la Universidad de Erlangen-Nuremberg (Alemania), en su sistema de simulación denominado VEINS.

2.3 Resumen

En este capítulo se ha presentado el concepto de red vehicular (VANET), se han mostrado la infinidad de aplicaciones de gran interés que puede tener, así como el auge en su estudio e investigación en los últimos años.

Una vez claro el concepto de VANET, así como la breve cronología de su desarrollo y los requerimientos tecnológicos que conlleva, se ha expuesto la necesidad de la simulación. Como se ha visto, las simulaciones juegan un papel fundamental en la validación de los protocolos de comunicaciones que es necesario desarrollar para este

tipo de sistemas. En este sentido, se ha hablado de los simuladores de red y los simuladores de tráfico (modelos de movilidad), así como de la necesidad de una interacción entre ambos, para sí lograr una simulación de verdad realista.

En los próximos capítulos, se verá qué simuladores existen y qué aportan, para poco a poco enfocarnos hacia el objetivo final de esta memoria, que es el simulador VEINS.

Capítulo 3

Simuladores de VANETs: Estado del arte

3.1 Introducción

Como ya se ha comentado en el capítulo anterior, en el contexto del desarrollo de las VANETS y sus sistemas de comunicación, la posibilidad de realizar simulaciones, sin tener que llevar a cabo experimentos de campo, permite llegar a soluciones que no impliquen inversiones a ciegas, ni construir nuevas y costosas infraestructuras. Los resultados obtenidos de las simulaciones se utilizan para poder convertir en realidad ideas tales como disminuir la congestión de tráfico, mejorar los servicios informativos para los conductores o proporcionar aplicaciones que permitan mejorar la seguridad en las carreteras.

Históricamente, el estudio de la movilidad ha sido un punto conflictivo en lo referente a la obtención de un modelo realista del movimiento de los vehículos. Además, como se ha anunciado, los modelos de movilidad están obligados a ser dinámicamente reconfigurables con el fin de reflejar los efectos de un protocolo de comunicación en particular.

En definitiva, para el desarrollo de los sistemas de simulación de redes vehiculares se hacen necesarios distintos protocolos, como *wireless multihop routing* y *broadcast* y, al mismo tiempo, son precisas herramientas que proporcionen movilidad a los vehículos ajustándose en lo más posible a la realidad. En este sentido, como se ha visto, la comunidad científica se encuentra trabajando en el desarrollo y/o en la reforma de modelos de movilidad específicos para vehículos, así como en sistemas de modelado del tráfico de datos.

En este capítulo se introducirán algunas de las soluciones disponibles, es decir, distintas formas de afrontar el problema de la simulación de las redes vehiculares, variando desde la más trivial hasta la más realista.

Pese a que la falta de coordinación, debido a que cada grupo de investigación desarrolla un modelo bajo sus propias necesidades, ha provocado la ralentización de este proceso, se han producido mejoras tanto en los modelos de movilidad de tráfico rodado, como en las herramientas para su estudio, desencadenado una investigación más profunda de escenarios más complejos y cada vez más grandes, así como un análisis más preciso de protocolos VANET y sus aplicaciones.

A continuación se introducirán las herramientas para la simulación vehicular disponibles en la comunidad investigadora, haciendo especial hincapié en la clasificación de simuladores de código abierto que se nos ofrecen.

3.2 Taxonomía de los simuladores

Previamente se ha indicado que las VANETs suponen un caso particular de las MANETs, caracterizadas por muy altas velocidades y un muy limitado grado de libertad en los patrones de movimiento de los nodos. Por lo tanto, un aspecto crítico en la simulación es la necesidad de un modelo de movilidad que refleje el comportamiento real del tráfico de vehículos. Además, la necesidad de crear escenarios donde la comunicación vehicular pudiera alterar la movilidad, y donde la movilidad mejorara las capacidades de la red, lleva a la necesidad de interacción entre un modelo de movilidad y un simulador de red, algo que en sus inicios no existía.

A continuación se muestran los diferentes enfoques para lograr el objetivo final: simular redes vehiculares (ver Figura 11) [Har09].

A. Simuladores Aislados

Originalmente se buscaba la forma de proporcionar control a los simuladores de red sobre los patrones de movilidad. Por ello, se les dotó con la posibilidad de cargar patrones de configuraciones óptimas estáticas de movilidad. Dado que dichos patrones deben generarse previamente a la simulación, no son posibles modificaciones y no existe interacción entre los dos mundos. La mayoría de los recientes generadores de movilidad a disposición del usuario pertenecen a esta categoría. Cabe destacar que la comunidad investigadora se ha adaptado al hecho de que los mundos sean aislados, ya que este hecho ha permitido su evolución independiente.

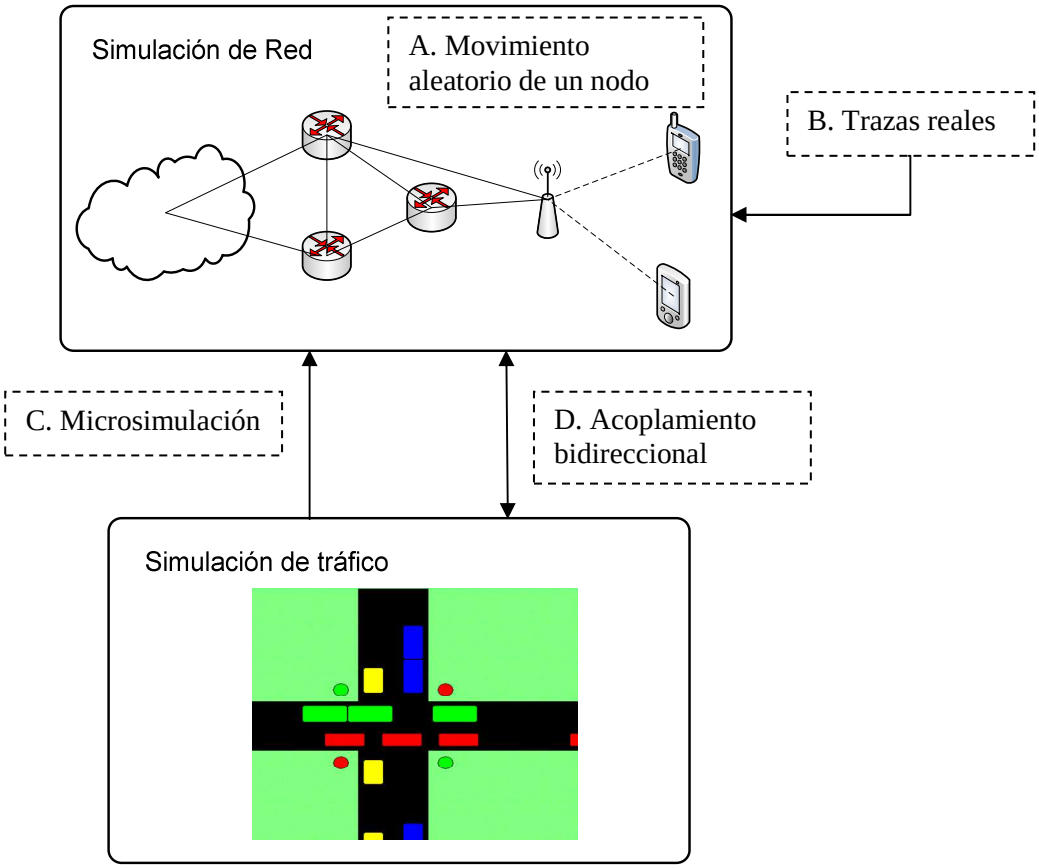


Figura 11. Técnicas para la simulación de redes vehiculares

B. Simuladores Integrados

De cara a solucionar el problema de la interacción entre el modelo de movilidad y el simulador de red, se propone sustituir ambos por un solo simulador de eventos de carácter más simplista. De esta manera se consigue la comunicación entre el modelo de movilidad y de red, pero como contrapartida, se hace uso de unos sistemas menos elaborados.

En definitiva, esto implica que el simulador de red sea de mala calidad, ya que los últimos protocolos de enrutamiento *ad-hoc* móviles necesitan capas MAC y físicas realistas. Como nota adicional, cabe indicar que la principal tendencia hoy en día en desarrollo por parte de los simuladores de redes, es un casamiento entre los protocolos estándares y la eficiencia computacional a través de la computación paralela y distribuida.

C. Simuladores Híbridos

Constituyen la unión híbrida, mediante una interfaz, entre los simuladores de red y los de movilidad. La dinámica de trabajo consiste en la alteración de los patrones de movilidad y red (y al contrario), provocando su funcionamiento en paralelo. Sin embargo, a pesar de que esta técnica consigue que nuevos modelos de movilidad se adapten a trabajar con simuladores de red modernos y eficientes, la carga computacional requerida puede ser considerable.

3.3 Herramientas de código abierto

En la actualidad existen numerosas herramientas de código abierto a nuestra disposición. A continuación se hará un breve recorrido por éstas, en primer lugar, diferenciando entre las herramientas de movilidad y los simuladores de red, para finalizar en aquellas de tipo integrado e híbrido (simuladores de VANETs propiamente dichos).

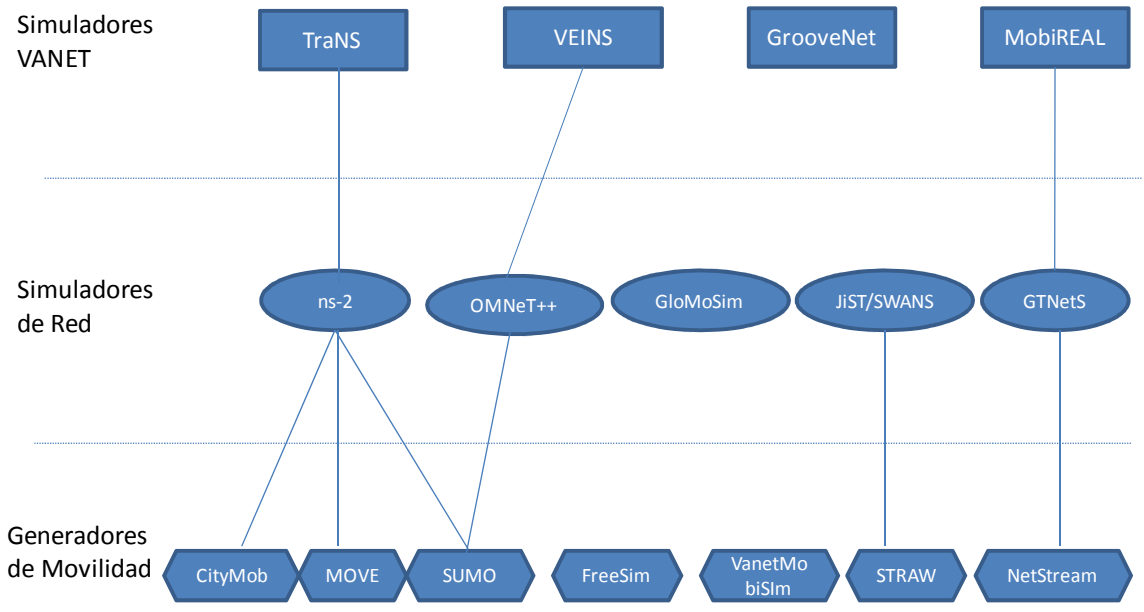


Figura 12. Taxonomía de los simuladores de redes vehiculares (algunos de código abierto)

La Figura 12 muestra la clasificación de algunos de los diferentes tipos de simuladores que serán descritos a lo largo de esta sección.

3.3.1 Simuladores de Tráfico de vehículos

Su desarrollo tuvo lugar con el auge del modelado de puntos conflictivos de autopistas o intersecciones de carreteras urbanas. No obstante, de cara a su utilización para la simulación de las redes vehiculares, estos simuladores presentan varias limitaciones. La primera se debe a que por el hecho de que fueran creadas de cara al transporte y al tráfico, su diseño no estaba basado en la generación de trazas de movilidad para simuladores de red.

La segunda limitación radica en la complejidad de su calibración. Debido al gran nivel de realismo que se da en la planificación del tráfico y del transporte, los usuarios precisan ajustar un gran número de parámetros, cada uno de ellos con una influencia determinada. Esto resulta ineficiente, ya que las simulaciones de VANETs no parecen requerir ese nivel de detalle para la movilidad vehicular.

Algunas de las herramientas para la generación de movilidad en código abierto son:

- **TEXAS (Traffic EXperimental and Analytical Simulation)**

Se trata de un modelo de simulación de tráfico en intersecciones, cuyo código está disponible en internet y puede emplearse tanto para la evaluación previa de nuevos diseños, como para simular los efectos de situaciones como cambios en el entorno de las intersecciones. Los resultados obtenidos implican un ahorro de recursos comparado con los actuales estudios de campo, y la verisimilitud de los datos ha sido contrastada.

- **SUMO (Simulation of Urban MObility)**

Permite la simulación en grades redes de carreteras. El generador de tráfico tiene la opción de importar topologías de diferentes fuentes, incluso crearlas desde cero. Utiliza el modelo de *coche seguidor*. Las trazas de salida pueden ser utilizadas directamente por simuladores de redes. Se estudiará en mucha más profundidad en los siguientes capítulos.

- **FreeSim**

Programado en Java, este simulador permite la representación y carga de escenarios como estructuras gráficas de múltiples sistemas de carreteras. Acepta crear algoritmos para el tráfico y aplicarlos a toda la red, un nodo o a un vehículo. Los datos de entrada pueden ser creados por el usuario, o importados de alguna organización del transporte.

- **STRAW (STreet RAndom Waypoint)**

Contiene un bloque complejo de constantes de movimiento y de datos de topologías urbanas importadas. El simulador de tráfico se basa en la variante del modelo de *coche seguidor* de Nagel-Schreckenberg.

- **Monarch (MObile Networking ARCHitectures)**

Se trata de una herramienta que extrae topologías de mapas reales obtenidos de la base de datos TIGER [USCB], cuyo modelo de movilidad es del tipo *Random way point*. Existe la posibilidad de extraer trazas útiles para ser introducidas en ns-2, lo que produjo que este simulador de red fuera extendido para ser usado con redes inalámbricas, convirtiéndose en la referencia para las redes MANET.

- **VanetMobiSim**

Permite importar mapas de la base de datos TIGER y ofrece la posibilidad de crear topologías mediante los grafos de Voronoi [Zim05]. En el nivel microscópico incluye modelos de movilidad como *Modelo de conductor inteligente* (con o sin cambios de carril), junto con gestión de intersecciones. Este modelo se basa en Java y puede generar trazas para luego ser utilizadas por diferentes simuladores de red como ns-2.

- **CityMob**

Es un simulador diseñado para explotar los diferentes modelos de movilidad en VANETs, con una clara vocación a utilizarse como herramienta para una posterior medida del impacto en el rendimiento de la comunicación intervehicular. Crea escenarios de movilidad urbana y simula coches accidentados, de tal forma que las trazas generadas pueden alimentar al simulador de redes ns-2 [NS2].

- **MITSIMLab**

Esta herramienta, programada en C++, fue desarrollada para evaluar los impactos de los distintos escenarios del sistema de gestión de tráfico, y así poder refinar su diseño. Dispone de una amplia gama de modelos de respuesta de los conductores a la información del tráfico en tiempo real, así como la interacción dinámica entre el sistema de gestión de tráfico y los controladores en la red.

- **Mobitools**

Se trata de una suite de movilidad desarrollada en Java, que utiliza mapas reales para la creación de trazas de movilidad, el visualizador MobiView (aplicación que permite acceder a las imágenes Landsat de la NASA), y un módulo para calcular los efectos de la transmisión. De igual modo, es compatible con ns-2, además de con Google Maps y con TIGER.

- **Translite**

Herramienta basada en SUMO y programada en Java. Hace uso de TIGER, OpenStreetMap (del que se hablará más en el capítulo 6) y Google Earth. Es capaz de generar trazas de movilidad para ser insertadas en ns-2.

Se puede describir como una versión básica de la utilidad TraNS, de la que se hablará a continuación.

3.3.2 Simuladores de Redes de comunicaciones

Al igual que se han presentado algunos de los más importantes simuladores de tráfico de código abierto, a continuación se describen los simuladores de red de tipo *freeware*.

- **NS-2**

Se trata del simulador de redes más conocido [NS2] debido a su modularidad y su universalidad. Como se ha comentado previamente, el proyecto Monarch amplió su horizonte hacia las redes inalámbricas.

Se estructura con las principales capas de la pila OSI. Los principales protocolos de MANETs y las capas físicas MAC han sido recientemente extendidas y validadas para el modelado de redes vehiculares.

- **GloMoSim**

Supone una alternativa a ns-2, siendo la versión gratuita de QualNet, pero con capacidades reducidas debido a la menor penetración de esta herramienta en la comunidad científica, lo cual provoca que el conjunto de protocolos disponibles sea muy reducido.

- **JIST-SWANS**

Este simulador escalable, basado en la plataforma JiST y programado en Java, a pesar de su poca popularidad inicial, ha conseguido llegar a una gran comunidad, pues dispone del modelado de los principales protocolos de MANETs. JiST es una herramienta de altas prestaciones, de eventos discretos que corren en la máquina virtual de Java. Su principal ventaja es que, además de proporcionar las mismas prestaciones que ns-2 y GloMoSim, permite simular redes con una mayor cantidad de nodos.

- **GTNetS**

Permite estudiar el comportamiento de redes de moderada a gran escala. Se encuentra en estricta conformidad con las distintas capas de los protocolos, lo que lo hace una solución más sencilla a la hora de desarrollar protocolos para MANETs [GTNETS].

- **OMNeT++**

Es una herramienta extensible y modular, basada en C ++ y dedicada principalmente a la creación de simulaciones de red. La palabra red se entiende en un sentido más amplio, ya que incluye no sólo redes cableadas e inalámbricas, sino redes dentro del chip, redes de colas, redes de sensores, redes inalámbricas *ad-hoc*, redes fotónicas, etc. OMNeT++ es libre para uso académico y es ya una plataforma ampliamente utilizada por la comunidad científica mundial. Se discutirá en mucha mayor profundidad sobre este simulador en los siguientes capítulos.

- **SNS (Staged Network Simulator)**

Desarrollado bajo la técnica de simulación por etapas, lo que permite eliminar cálculos redundantes a través de la función de almacenamiento en caché, permitiendo así una mayor velocidad y en escalabilidad. Se trata de una herramienta basada en ns-2.

3.3.3 Simuladores de VANETs Integrados

En esta sección veremos las herramientas para simulación de VANETs de tipo integrado que están disponibles de manera abierta a la comunidad investigadora. Tal y como se describe en el apartado B de la sección 3.2, estos sistemas integrados se conforman mediante un único modelo que contempla tanto la movilidad, como la simulación de red.

- **NS3**

Se trata de la próxima generación del popular ns-2. Resulta de la integración de un modelo de movilidad junto con una aplicación para autovías [Arb10], lo que lo convierte en una herramienta interesante, pero con prestaciones alejadas de las de otros sistemas más complejos. La movilidad de vehículos y comunicación en red se maneja a través de eventos.

- **NCTUns**

Realmente debería definirse como un simulador de red extensible, que además de ser capaz de simular distintos protocolos (tanto en redes cableadas como en inalámbricas), contiene alguna funcionalidad para el modelado de la movilidad de vehículos, tales como patrones de conducción humana, *coche seguidor* y control de intersecciones. La parte de red contempla la capa de acceso al medio IEEE WAVE 802.11p para entornos vehiculares.

3.3.4 Simuladores de VANETs Híbridos

Como ya se ha indicado, los simuladores integrados ofrecen características avanzadas de cara al modelado del movimiento de vehículos y capacidades de red, pero quedan limitados por su compleja configuración y sus bajas prestaciones. De este modo, se busca asociar los simuladores de red y movilidad existentes, creándose así los simuladores híbridos de VANETs.

- **MobiREAL**

Este nuevo simulador (desarrollado en C), que contempla modelos para dispositivos móviles, es capaz de representar la movilidad real de personas y vehículos, cambiando su comportamiento en función de una determinada aplicación VANET, y obteniendo una evaluación detallada de las aplicaciones de red, protocolos de enrutamiento, infraestructuras, etc.

MobiREAL permite simular una amplia variedad de redes móviles *ad-hoc*, mediante la adición de modelos de movilidad al simulador de redes GTNetS. Por otro lado, MobiREAL *Animator* permite visualizar el movimiento del nodo, la conectividad, y la transmisión de paquetes, ayudando así a la comprensión de los resultados.

Para la movilidad vehicular, se utiliza la modificación de esta herramienta que incluye un simulador de tráfico llamado NetStream. Sin embargo, el inconveniente de no tratarse de un software libre, limita su uso.

- **TraNS**

El enfoque de esta herramienta, desarrollada en Java, se sustenta en la unificación del generador de movilidad SUMO con el simulador de redes ns-2. A través del uso del denominado interfaz *Interpreter*, ambos simuladores se interconectan. Las trazas de movilidad de SUMO son transmitidas a ns-2, y del mismo modo, las instrucciones de ns-2 se envían a SUMO.

Se trata de un sistema de amplio uso y gran popularidad, pues logra resultados interesantes. A parte de lograr el acoplamiento entre los dos mundos, gracias a SUMO, las trazas de movilidad vehicular utilizadas son de gran realismo.

- **VEINS (VEhicles In Network Simulation)**

Está compuesto por el simulador de redes basado en eventos OMNET++ y el generador de movilidad microscópico SUMO, trabajando acopladamente y en tiempo real.

Hablaremos más detenidamente de esta herramienta en los subsiguientes capítulos de esta memoria. No obstante, habiendo hecho este análisis de la taxonomía de los simuladores, cabe mencionar la ventaja que aporta sobre otros sistemas similares, concretamente, sobre TraNS.

En definitiva, TraNS logra la interconexión entre SUMO y ns-2 mediante el uso de un bucle activo por el que el simulador de red envía comandos de control al de tráfico. Esto permite detener a los vehículos y/o modificar sus rutas. Sin embargo, en TraNS no se ha considerado el control del tiempo de simulación. VEINS contribuye al estado del arte de los simuladores de VANETs con una muy minuciosa interfaz de control entre ambos dominios de simulación. A esto se debe unir el hecho de que el proceso de desarrollo de TraNS se ha visto discontinuado en los últimos tiempos.

3.4 Resumen

En el capítulo 3 se ha realizado un estudio en profundidad del estado del arte de los sistemas de simulación para redes vehiculares, dirigiendo una especial atención a los de código abierto.

Llegados a este punto, el lector conoce las utilidades de las VANETs, así como es consciente de la necesidad de las simulaciones de cara al desarrollo de las mismas. Ahora también sabe qué tipos de herramientas hay a su disposición para este fin.

Como se ha explicado, el mundo de la simulación vehicular parece converger hacia soluciones híbridas, teniendo a TraNS y VEINS como su máxima expresión. Ambos hacen uso del simulador de tráfico SUMO, mientras que para la simulación de red, el primero utiliza ns-2, y el segundo OMNeT++.

Por las ventajas que aporta, a partir del siguiente capítulo, esta memoria se centrará en el estudio de VEINS y sus componentes. Se analizará en profundidad su sistema de funcionamiento y prestaciones, así como se facilitará una guía para su uso.

Capítulo 4

Simulador VEINS

4.1 Introducción

El presente capítulo se centrará en el análisis de las bases de funcionamiento y prestaciones que, en concreto, el entorno de simulación VEINS nos ofrece.

En los siguientes apartados se estudia en detalle cómo se lleva a cabo la simulación de redes vehiculares mediante VEINS. En concreto, la solución propuesta hace uso de dos importantes sistemas de simulación de código abierto: SUMO (simulador de tráfico) y OMNeT++ (simulador de red), introduciendo un módulo de comunicación entre ambos que trabaja como una extensión de SUMO, denominado TraCI. Por lo tanto, el capítulo está estructurado, por simplicidad, analizando por orden cada uno de los bloques que conforman el entorno VEINS. Es decir, en primer lugar se estudia el simulador de tráfico SUMO, posteriormente el simulador de red OMNeT++ y, finalmente, se presentará una atención especial a lo que aporta el módulo de comunicación entre ambos.

Durante el trascurso de los distintos apartados se verá la potencia de las prestaciones de estos simuladores de código abierto y las enormes ventajas de cara al realismo de la simulación global de la red vehicular, que el módulo TraCI nos ofrece.

4.2 Simulador de tráfico: SUMO

Pese a que cabría pensar que el tráfico podría describirse mediante la hora de salida, la ruta seguida y una cierta duración, como se ha visto, esta aproximación es totalmente equivocada. El tráfico está fuertemente condicionado por el deseo de movilidad individual, por lo que ni los tiempos de salida son fijos, ni las rutas son prefijadas.

Esto supone, claramente, un gran problema a la hora de realizar un modelado del tráfico, especialmente en el caso de los recorridos privados. En este caso, la descripción del tráfico mediante el uso de fórmulas matemáticas resulta aún más complejo, o incluso imposible. Tanto el deseo humano de marcharse de un lugar e ir a otro distinto en unas horas determinadas, como el movimiento del vehículo en la calle, tienen una fuerte y clara influencia en el tráfico. Es más, ambos factores se influencian el uno al otro: la carga de la red viaria (las calles) depende de las horas de salida de los conductores y, a su vez, determina la velocidad del movimiento. Por otro lado, la carga de la red tiene un efecto en las horas de salida de los conductores, pues desean moverse rápidamente y tener certeza de a qué hora van a alcanzar su destino. A parte de esto, el tráfico se ve influenciado por otros parámetros, tales como las condiciones meteorológicas, la infraestructura presente u otros incidentes que puedan estar afectando al sistema.

Con la propuesta de lograr una simulación que tuviera en cuenta todos estos factores, se desarrolló el proyecto SMARTEST, que fue una iniciativa colaborativa entre distintos organismos y cofinanciada por la Comisión Europea [SMARTEST]. Es en el marco de este proyecto, o mejor dicho, tras las conclusiones obtenidas en el mismo, cuando el *Instituto de Investigación en el Transporte* (*Centro Aeroespacial Alemán*), junto con el *Centro de Informática Aplicada* de Colonia (Alemania), decide desarrollar una herramienta de código abierto, que satisficiera las necesidades descubiertas.

Es así como en el año 2000 nace SUMO [Kra02], que es el acrónimo de *Simulation for Urban MObility* (*Simulación para Movilidad Urbana*). Se trata de un paquete de software de código abierto para simulación de tráfico microscópico y multimodal.

La primera cualidad da idea de que el modelo de movilidad considera cada vehículo como una entidad distinta, lo que incrementa su carga computacional (en contraposición con un sistema macroscópico, que modelaría únicamente cantidades de interés, en términos brutos, como por ejemplo velocidad media o densidad vehicular).

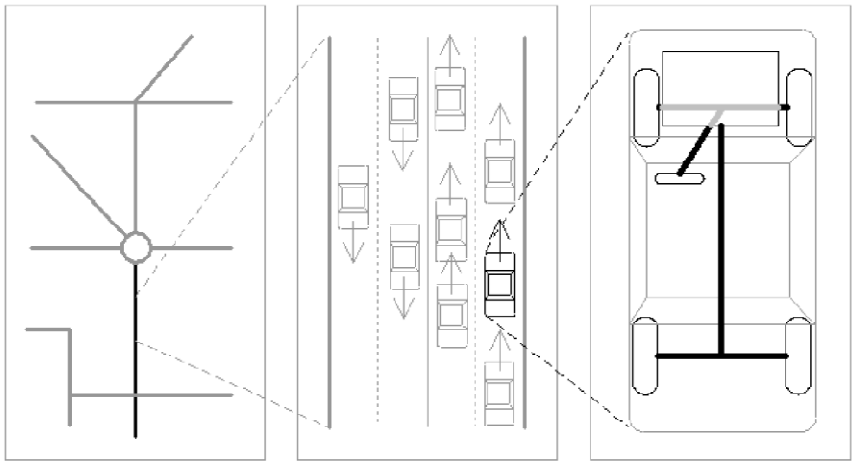


Figura 13. (De izquierda a derecha) simulación macroscópica, microscópica (caso de SUMO) y submicroscópica [Kra02]

4.2.1 Paradigmas básicos

SUMO está concebido para simular tráfico de una red viaria del tamaño de una ciudad, no obstante, también es capaz de modelar redes de tamaños mayores, tales como sistemas de autopistas. Puesto que la simulación es multimodal, lo que significa que no sólo se modelan los movimientos de los coches, sino también los sistemas de transporte público, la componente más básica de la simulación es un ser humano individual. Este ser humano es descrito mediante una hora de salida y la ruta que va a tomar (que estará formada por subrutas, que describen una única modalidad de tráfico).

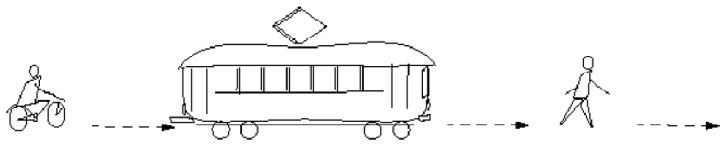


Figura 14. Simulación de tráfico multimodal

Esto implica que, por ejemplo, en la simulación una persona pueda ir en su coche hasta la estación más cercana de la red de transporte público y continuar desplazándose mediante otro medio de transporte. No obstante, el desplazamiento a pie no se contempla en la simulación, pero el sistema sí que estima el tiempo de que precisa el peatón para completar la ruta. Ésta última funcionalidad, sin embargo, carece de interés para el ámbito de las VANETs.

Como ya se ha dicho, SUMO simula el flujo de tráfico de forma microscópica. Esto significa que cada vehículo que se mueve por la red simulada es modelado individualmente y está caracterizado por una cierta posición y velocidad. Por cada intervalo temporal, que tiene duración de 1 segundo, se actualizan estos parámetros dependiendo del vehículo que esté situado delante y de la topología de la vía por la que se desplaza. La simulación de los vehículos se hace de forma discreta en el tiempo y continua en el espacio, pues el modelo de movilidad que utiliza SUMO es continuo.

4.2.2 Características

SUMO está en constante evolución y desarrollo, pues se trata de un sistema de código abierto. No obstante, para la versión utilizada durante el desarrollo de este proyecto (que es la 0.12, aunque más adelante se hablará más en detalle de esto), las funcionalidades de que dispone el simulador pueden resumirse a grandes rasgos en los siguientes puntos principales [SUMO]:

- Movimiento de vehículos en tiempo discreto y espacio continuo.
- Soporta tipos diferentes de vehículos.
- Vías multicarril.
- Cambio de carril.
- Diferentes modos de prioridad en cruces y semáforos.
- Interfaz gráfica.
- Soporta redes viarias de varias decenas de miles de calles.
- Alta velocidad de ejecución (alcanza 100.000 actualizaciones/segundo por vehículo en un procesador de 1GHz).
- Interoperabilidad con otras aplicaciones en tiempo real.
- Soporta importación de mapas topológicos para la red vial.
- Rutas microscópicas (cada vehículo tiene la suya propia).
- Alta portabilidad (paquetes para Linux y Windows).
- Alta interoperabilidad gracias al uso de datos XML.

4.2.3 Modelo de movilidad

En SUMO se utiliza una extensión del denominado modelo de *Gipps* (inventado y descrito por Krauß en [Kra98]). En este modelo, por cada instante de tiempo la velocidad del vehículo es adaptada a la del que va por delante, de tal modo que la simulación da

lugar a comportamiento exento de colisiones para los siguientes intervalos de tiempo. Esta velocidad v_{Segura}es calculada por el algoritmo de la siguiente forma:

$$v_{Segura}(t) = v_P(t) + \frac{g(t) \cdot t - v_P(t)}{\frac{\bar{v}}{b(\bar{v})} + \tau}$$

Donde:

- $v_P(t)$ es velocidad del vehículo precedente en el instante t.
- $g(t)$ es la distancia o *gap* al vehículo precedente en el instante t.
- τ es el tiempo de reacción del conductor (normalmente 1s).
- b es la función de deceleración.

Si nos fijamos en la clasificación que se hizo de los modelos de movilidad en el capítulo 2, el utilizado en SUMO encajaría claramente en la categoría de modelo sintético de *coche seguidor*. Podemos ver en el diagrama de la Figura 15, un esquema del principio de funcionamiento de este modelo.

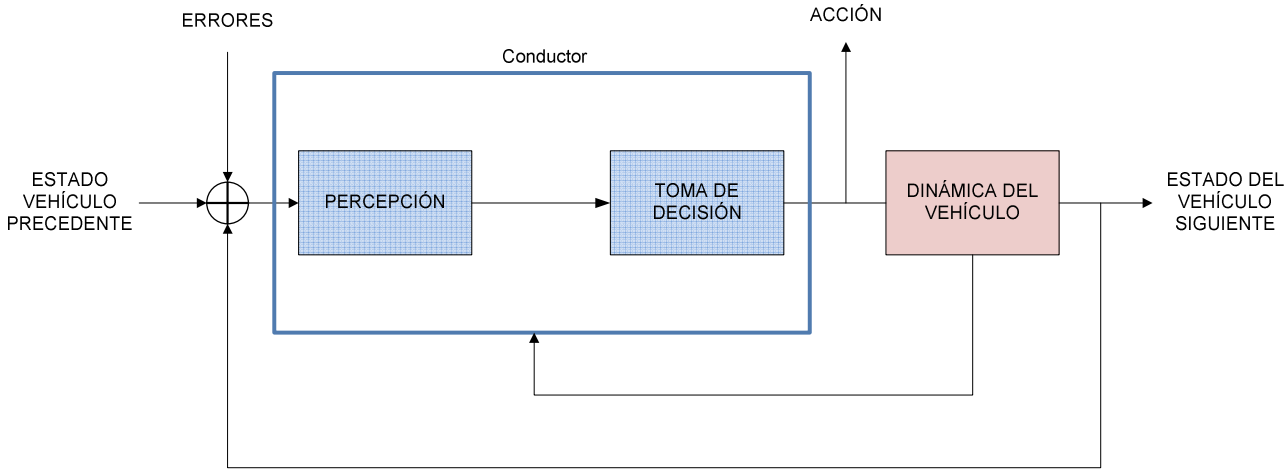


Figura 15. Diagrama del modelo de movilidad de coche seguidor utilizado en SUMO

Además, prestando atención a los componentes más importantes del modelo de movilidad (descritos en la sección 2.2.4, ver Figura 10), cabría mencionar los siguientes:

Mapas topológicos

- Definidos por el usuario de forma manual, es decir, definiendo cruces y puntos de unión entre ellos (las vías o calles), uno a uno.
- Aleatorios.
- Mapas reales mediante importación.

Origen/Destino

Igualmente, pueden ser definidos por el usuario, o ser aleatorios.

Gestión de las intersecciones

Los semáforos juegan un importante rol en la gestión de las intersecciones. A parte de implementarse reglas básicas de prioridad en cruces, cada intersección en la simulación puede también estar controlada mediante semáforos.

Generador de tráfico

Es el responsable de la "producción" de tráfico. Es aquí donde se implementa el ya mencionado paradigma del *coche seguidor*. Como veíamos en el diagrama de bloques de la Figura 15, el modelo considera que el conductor está sometido a la interacción con el resto de vehículos, y su comportamiento simulado se asemejará al de un ser humano.

En lo que se refiere a la generación de las rutas, el modelo no distribuye los vehículos estadísticamente por la red, sino que utiliza los datos del plan diario de los conductores, en términos de recorrido y horas de salida. Estos datos son cargados en el simulador, no obstante, SUMO debe calcular las rutas en sí, a partir de los mismos. Existe un módulo específico del simulador que es el encargado de leer estos tiempos de partida, puntos de origen y destino, y calcula las rutas mediante el algoritmo de Dijkstra (1959).

Puesto que la velocidad en las calles varía con la cantidad de tráfico, el cálculo de las rutas previo a conocer el tráfico puede no ajustarse a una situación real. En este caso, la obtención de las rutas se realiza mediante el algoritmo de *Equilibrio de Usuario Dinámico* (*Dynamic User Equilibrium*), desarrollado por Christian Gawron [Gaw98], donde el enrutamiento y la simulación se repite varias veces para lograr un comportamiento de los conductores similar al del mundo real.

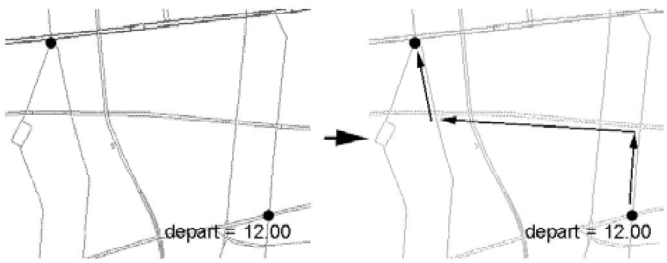


Figura 16. Cálculo de la ruta en una red pequeña

Para finalizar, cabe mencionar que SUMO es un simulador validado, esto quiere decir que los patrones de movilidad que genera han sido comparados con trazas reales y ha sido medido el error que se produce en la simulación.

4.3 Simulador de red: OMNeT++

OMNeT++ es un simulador de eventos discretos basado en C++ para el modelado de redes de comunicaciones, multiprocesadores y otros sistemas distribuidos. La motivación de desarrollar OMNeT++ surgió de la necesidad de una herramienta potente para simulación basada en eventos discretos, que fuera de código abierto, para su uso en el ámbito académico, educativo y en instituciones comerciales orientadas a la investigación.

Esta herramienta está disponible al público desde septiembre del año 1997, y ha ido ganando popularidad en el trascurso de su desarrollo durante estos años. Se trata de un proyecto surgido en el Departamento de Telecomunicaciones de la Universidad de Tecnología y Economía de Budapest (Hungría) [Var01].

4.3.1 Estructura modular

Un modelo OMNeT++ está compuesto por módulos jerárquicos, que se comunican entre sí mediante paso de mensajes. Los módulos básicos, denominados *módulos simples*, se agrupan en *módulos compuestos*, los cuales pueden agruparse nuevamente con otros módulos compuestos, y así sucesivamente (y sin limitación), estando todos ellos contenidos en el de nivel superior, llamado *módulo de sistema*. Se puede ver un esquema en la Figura 17.

Todos estos módulos simples son instancias de un módulo básico llamado *módulo "type"*, que es quien provee la funcionalidad básica para implementar los módulos que compondrán el modelo.

Debido a la estructura jerárquica de este sistema, los mensajes (que pueden contener estructuras de datos) típicamente viajan a través de cadenas de conexiones entre módulos, para acabar siempre en uno simple. Los interfaces de comunicación entre los módulos se denominan *puertas*, que a su vez están conectadas unas con otras mediante *conexiones*.

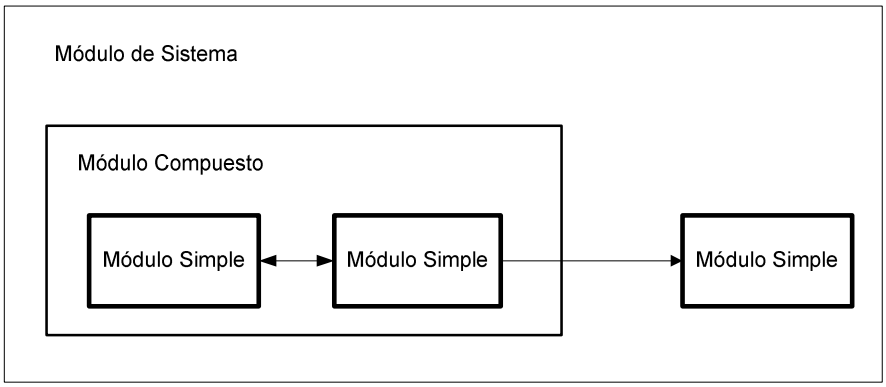


Figura 17. Estructura modular en OMNeT++

Cada conexión tiene tres parámetros asociados, que sirven para facilitar el modelado de la red de comunicaciones. Éstos son: el retardo en la propagación, la tasa de error de bit y la tasa de transferencia de información. De este modo se consigue personalizar el comportamiento de cada módulo y su interacción con los demás.

4.3.2 Paradigmas de la simulación

OMNeT++ ofrece una extensa colección de librerías de simulación, así como provee al desarrollador de las rutinas que servirán para controlarlas y de una interfaz de usuario que permite la creación de modelos y la ejecución y depurado de las simulaciones.

Un modelo OMNeT++ es básicamente una descripción de la topología que se desea simular, en el denominado lenguaje NED. Se trata de un lenguaje propio de OMNeT++ que permite definir los módulos, puertas y conexiones presentes en nuestra topología. Se verá una explicación más detallada de NED en el capítulo 6, donde se presentará una guía práctica de uso.

Un modelo de simulación en OMNeT++ consta de las siguientes etapas:

- Definición de la topología y estructura a simular, es decir, módulos e interconexiones, mediante el mencionado lenguaje NED.
- Definición de los módulos simples, que son los elementos activos del modelo, mediante C++.
- Compilación de los módulos y enlazado con la librería de simulación: Generación del modelo (ver Figura 18 a continuación).
- Especificación de los parámetros concretos de la simulación.

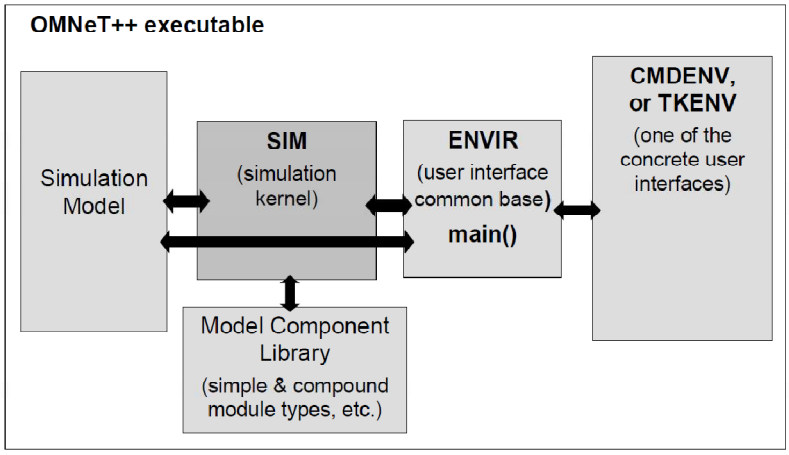


Figura 18. Arquitectura lógica de una simulación en OMNeT++ [Var01]

Cabe entrar a analizar qué componentes aporta la librería de simulación de OMNeT++. Su intención es la de cubrir de la forma más extensa posible las tareas más comunes de una simulación.

Dispone de generación de números aleatorios para diferentes distribuciones estadísticas; ofrece creación de colas (también con prioridad) y otras clases de contenedores de datos; aporta el objeto *mensaje*, que servirá para intercambiar información entre los distintos módulos y que puede contener estructuras de datos u otros objetos; soporta tráfico de datos de enrutamiento; etc.

4.3.3 Paquete INET

El marco de simulación INET se construye sobre la plataforma OMNeT++ y hace uso del mismo modo de funcionamiento, es decir, módulos que se comunican entre sí mediante paso de mensajes.

Se trata de un paquete de código abierto para simulación que contiene la implementación de los protocolos IPv4, IPv6, TCP, SCTP, UDP, algunos modelos de aplicación, así como un modelo de MPLS [INET].

No obstante, la funcionalidad de INET en la que nos centraremos es en la posibilidad que ofrece de simulación de redes móviles e inalámbricas.

Es obvio que en el ámbito de la simulación de las redes vehiculares, se precisa de un paquete como este, que aporte la componente inalámbrica al escenario de simulación.

4.3.4 MiXiM

Como se ha visto, INET está enfocado hacia el modelado de las capas superiores, es por ello que se precisa de otro bloque que aporte la funcionalidad de las inferiores. MiXiM es el marco encargado de esto: aporta la capa física y de enlace, así como una interfaz para comunicación con el paquete INET.

MiXiM es paquete de OMNeT++, creado para el modelado de redes inalámbricas (fijas o móviles), lo que lo hace válido para redes de sensores inalámbricas, redes de área corporal, redes *ad-hoc* y redes vehiculares. Ofrece modelos detallados de propagación radio, estimación de interferencias, consumo de potencia por dispositivos de radiocomunicaciones, así como de protocolos MAC inalámbricos.

Dentro de este marco de simulación se construyen modelos propiamente del interés de la simulación de las redes vehiculares. Este es el caso de [Som11], que propone un modelo realista y computacionalmente eficiente para la comunicación radio en ambientes urbanos sobre IEEE 802.11p.

El modelo, que se usará en el marco de simulación VEINS, está basado en medidas reales usando dispositivos 802.11p/DSRC, con las que se logra estimar el efecto que los edificios y otros obstáculos, tienen sobre la comunicación intervehicular. En [Som11] se hace notar que los modelos puramente estocásticos no son suficientes para evaluar las comunicaciones radio, algo especialmente grave en aplicaciones relacionadas con la seguridad en el campo de las VANETs.

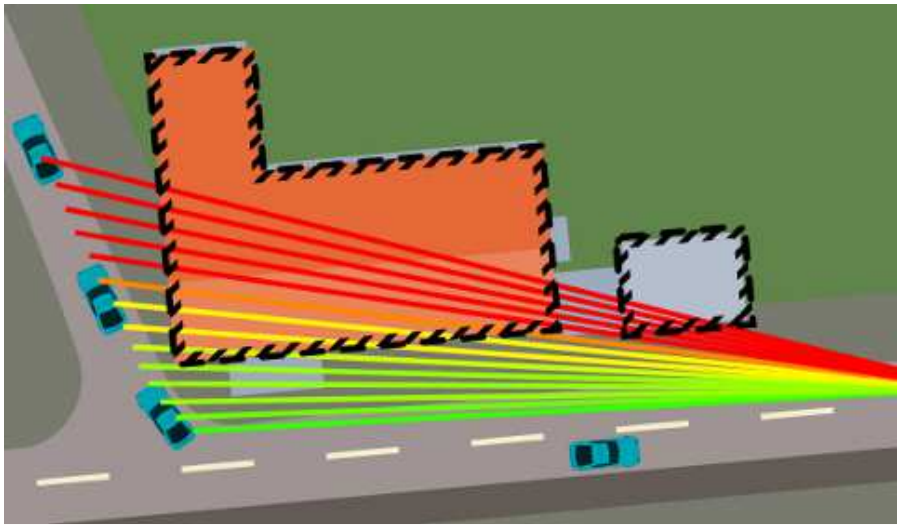


Figura 19. Mediante un módulo en la simulación de OMNeT++ se puede incluir un sencillo modelo de obstáculos para comunicaciones urbanas sobre 802.11p

4.4 Caracterización de VEINS

Pese a que quedó patente en la introducción, cabe hacer especial hincapié en que realmente durante todo el capítulo 4 se ha estado hablando del sistema VEINS. Hasta ahora no se ha hecho más que un estudio detallado de los pilares de este sistema, pues VEINS es un marco de simulación que hace uso tanto de OMNeT++ (y el paquete INET), como de SUMO, introduciendo un interfaz de comunicación entre ambos.

VEINS, que es el acrónimo de *Vehicles in Network Simulation* (*Vehículos en Simulación de Red*), surge ante la necesidad de un marco de simulación completo y de código abierto, en el que los resultados de la simulación de red fueran usados por la simulación de tráfico (y viceversa). Se trata de un proyecto construido en común entre la Universidad de Erlangen-Nuremberg (Alemania), Departamento de Redes de Ordenadores y Sistemas de Comunicación, y la Universidad de Innsbruck (Austria), Departamento de Ordenadores y Sistemas de Comunicación, con la colaboración del Centro Aeroespacial Alemán.

La primera versión vio la luz en el año 2008, pero desde entonces multitud de desarrolladores han colaborado para perfeccionar el sistema, lográndose las prestaciones de las que se hablará a continuación.

4.4.1 Simulación con acoplamiento bidireccional

Como ya se ha visto, tradicionalmente, los modelos de movilidad usados por muchas herramientas de simulación de red no tenían en cuenta el comportamiento del conductor, o las características específicas del entorno urbano. Como resultado de esto, la simulación de los protocolos de red para las VANETs era no realista.

Los modelos basados en trazas supusieron una importante ventaja en este sentido, pues generan patrones de movilidad realistas (de forma *offline*), que sirven para validar o evaluar protocolos de red. De hecho, es una práctica muy común en muchas plataformas de simulación que las trazas de movilidad generadas de forma independiente, se introduzcan en el simulador de red como ficheros *offline*. De esta forma se reduce la complejidad del sistema.

No obstante, una filosofía de diseño con tal “desacoplamiento” se enfrenta a un dilema fundamental (aunque quizás no inmediato):

Si los resultados del módulo de simulación de red pueden afectar a cómo se desarrollará el movimiento de los vehículos, precisaremos de un sistema de interacción en

tiempo real entre la simulación de movilidad y el módulo de simulación de red, ya que una metodología *offline* y aislada no es capaz de generar un resultado del todo realista.

Por ejemplo, en aplicaciones de seguridad vial, los vehículos generan mensajes de alerta con la intención de que el resto modifiquen sus patrones de movimiento. En este caso, el modelo de simulación de red debe interactuar con el modelo de simulación de movilidad en tiempo real.

VEINS nace con la idea de solucionar este problema, es decir, con el objetivo de lograr este acoplamiento bidireccional.

En el esquema que vemos a continuación (Figura 20) y que se explicará en más detalle en la siguiente sección, podemos ver cómo consigue VEINS el mencionado acoplamiento entre SUMO y OMNeT++.

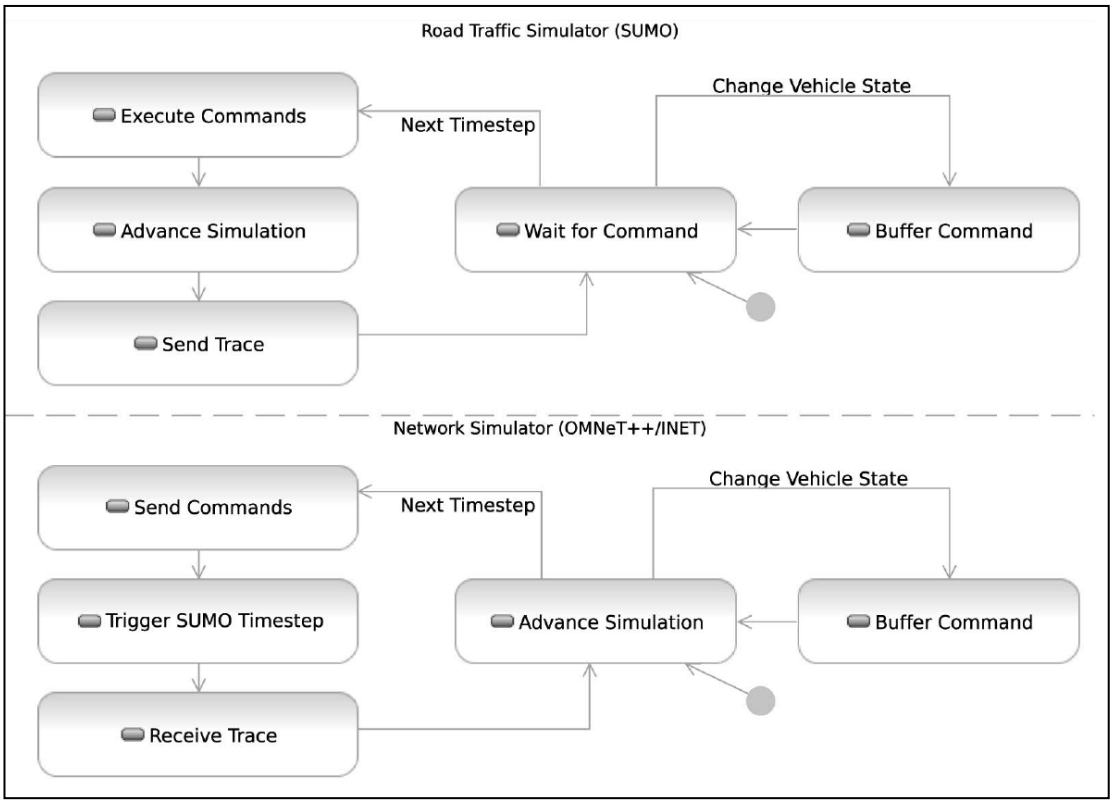


Figura 20. Visión general de la simulación bidireccional. Máquina de estados [Som08]

4.4.2 Sistema TraCI

La pregunta que surge ahora es cómo consigue VEINS este acoplamiento bidireccional. Veremos a continuación, y poco a poco, que la respuesta está en el denominado sistema TraCI.

VEINS aporta el acoplamiento entre ambos marcos de simulación, el de red OMNeT++ y el de tráfico SUMO, extendiendo cada uno de ellos con un módulo de comunicación dedicado. Cuando una simulación está teniendo lugar, estos módulos intercambian comandos, así como trazas de movilidad, a través de conexiones TCP.

TraCI, que es el acrónimo de *Traffic Control Interface* (*Interfaz de Control de Tráfico*), es la arquitectura de código abierto elegida por el marco de simulación VEINS como solución a este problema. Este sistema nace como un proyecto conjunto entre el Instituto de Telemática de la Universidad de Luebeck (Alemania) y el Laboratorio para Comunicaciones y Aplicaciones de la Escuela Politécnica Federal de Lausana (Suiza) en el año 2008 [Weg08]. Está diseñado para poder ser usado por diversos simuladores de tráfico y de red, encontrándose entre ellos, claro está, SUMO y OMNeT++. Como se verá a continuación y como ya se ha mencionado, el simulador de tráfico y de red se comunican mediante una conexión TCP en modo Cliente/Servidor (OMNeT++/SUMO), usando los denominados mensajes TraCI para intercambiarse comandos. Podemos ver el formato de estos mensajes en la Figura 21 a continuación.

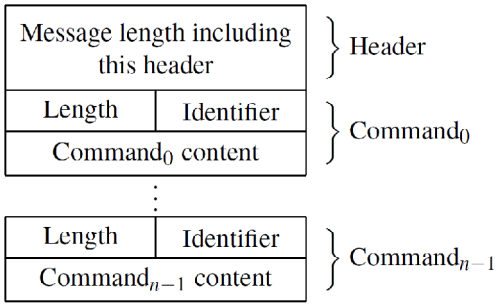


Figura 21. Formato de mensaje TraCI: Pequeña cabecera seguida de lista de comandos [Weg08]

OMNeT++ es un simulador basado en eventos, así que maneja la movilidad planificando el movimiento de los nodos en intervalos regulares de tiempo. Este modo de funcionamiento encaja muy bien con el de SUMO, pues éste también trabaja con instantes de simulación que avanzan en pasos discretos. Como se podía ver en la Figura 20 (máquina de estados), los módulos de control integrados en OMNeT++ y SUMO son

capaces de almacenar cualquier comando que les llegue entre intervalos de tiempo, lo cual garantiza una ejecución síncrona.

En cada instante de tiempo OMNeT++ envía todos los comandos almacenados en el búfer a SUMO y dispara la correspondiente iteración de la simulación de tráfico rodado. Una vez finalizada la simulación de tráfico para ese instante temporal, SUMO envía una serie de comandos, junto con la posición de todos los vehículos, de vuelta al módulo OMNeT++. Esto permite a OMNeT++ reaccionar en base a las trazas de movilidad recibidas, introduciendo nuevos nodos, eliminando algunos que ya hayan alcanzado su destino y, lo que es fundamental, moviendo el resto de nodos de acuerdo con lo que ha indicado su homólogo en la simulación de tráfico.

Tras procesar todos los comandos recibidos y haber movido todos los nodos de acuerdo con la información de movilidad procedente de SUMO, OMNeT++ avanza la simulación hasta el siguiente instante de tiempo, permitiendo a los nodos reaccionar ante las condiciones ambientales (que han sido alteradas), es decir, la comunicación intervehicular (IVC) está claramente influyendo en sus velocidades y rutas.

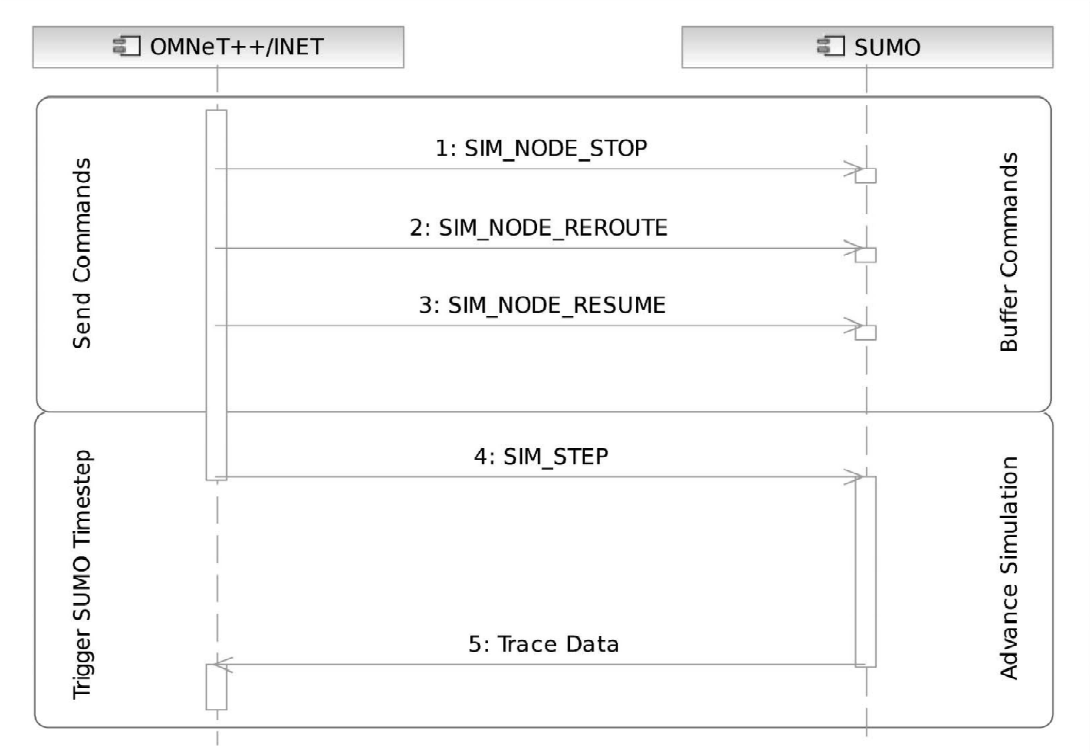


Figura 22. Secuencia de mensajes intercambiados entre los módulos de comunicación de los simuladores de red y tráfico [Som08]

En la Figura 22 anterior, se puede observar la interacción entre ambos simuladores, en forma de diagrama de intercambio de mensajes. Usando un simple protocolo de petición/respuesta, el tráfico rodado en SUMO puede verse influenciado por OMNeT++ de muchas formas. Es más, como vemos en el diagrama, es el propio OMNeT++ quien indica a SUMO que debe avanzar al siguiente instante de tiempo para la simulación.

Los vehículos pueden ser detenidos para crear atascos artificiales, puede hacerse que continúen su marcha, haciendo así que se finalice el atasco, y cada vehículo puede ser reorientado hacia un segmento de la carretera diferente y arbitrario. De esta forma, VEINS logra de forma precisa reflejar cómo un conductor que observa que una zona está atascada, trata de evitarla.

En el diagrama anterior (Figura 22) podemos ver las dos fases alternantes de la simulación bidireccionalmente acoplada. En la primera fase se enviarán los comandos a SUMO, mientras que en la segunda se forzará su ejecución para recibir la traza de movilidad resultante. Con este mecanismo, ambos simuladores se encuentran funcionando de forma altamente acoplada y SUMO sólo puede realizar un nuevo paso de simulación una vez que el simulador de red ha sido capaz de procesar todos los eventos correspondientes al instante actual. Hay que tener en cuenta que el simulador de red hace que la microsimulación de tráfico rodado avance en intervalos fijos de tiempo, lo cual significa que la granularidad de estos intervalos debe ser lo suficientemente fina para obtener resultados realistas. Esto no supone un problema, pues la microsimulación del tráfico que realiza SUMO puede ser procesada muy rápido en comparación con la simulación de una red inalámbrica.

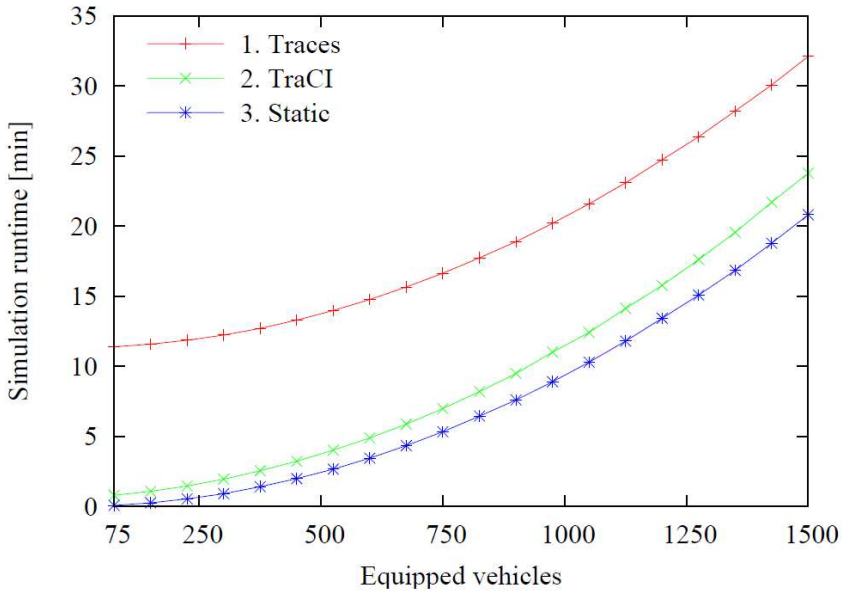


Figura 23. Comparativa del tiempo de ejecución usando o no TraCI [Weg08]

Se demuestra que el enfoque de TraCI puede competir perfectamente con la forma tradicional de funcionamiento de los simuladores desacoplados para VANETs, en lo que a tiempo de ejecución se refiere. Podemos ver una comparativa en la Figura 23 anterior, en la que se representa el tiempo total de ejecución en minutos, frente al número de vehículos simulados. En azul vemos la curva correspondiente a una situación en la que los nodos permanecen estáticos durante toda la simulación; la roja corresponde al caso tradicional en el que las trazas de movilidad son generadas de forma *offline* por SUMO, para posteriormente ser pasadas al simulador de red; y finalmente, la verde pertenece a la simulación realizada con acoplamiento bidireccional mediante TraCI.

Como vemos, para el caso en el que se introducen las trazas de modo *offline*, el tiempo total (es decir, ejecución de ambas simulaciones, una a continuación de la otra) es considerablemente mayor. Por otro lado, si comparamos la respuesta del sistema de simulación usando TraCI con el caso estático (que es la referencia, pues no hay simulador de tráfico interviniendo), se puede concluir que introducir movilidad y acoplar ambos simuladores mediante TraCI no supone un incremento importante en el tiempo de ejecución correspondiente únicamente al simulador de red.

La Figura 24 mostrada a continuación corresponde a un pequeño ejemplo de los comandos y trazas de movilidad enviados por SUMO al simulador de red. Para garantizar el correcto sincronismo de ambos sistemas, podemos ver que cada instante de simulación es señalizado mediante un comando *tsp* (*time syncrhonization protocol*) que contiene el momento actual de la simulación.

```
tsp 0
add host[0000] Car;i=car0_vs;r=0,,#707070,1
mov host[0000] 998.35  4995.00   0.00  0901
tsp 8
add host[0002] Car;i=car1_vs;r=0,,#707070,1
mov host[0002] 998.35  4993.42   0.00  0901
mov host[0001] 998.35  4976.32   6.74  0901
mov host[0000] 998.35  4943.28   9.83  0901
[...]
tsp 529
del host[0000]
mov host[0003] 3786.65  998.35  13.89  0404
mov host[0002] 3911.91  998.35  13.90  0404
mov host[0001] 3954.35  998.35  13.89  0404
```

Figura 24. Extracto de una traza de movimiento enviada por el simulador de tráfico [Som08]

Mediante el comando *add*, la simulación de tráfico es capaz de introducir nuevos vehículos. En el ejemplo, se introducen nuevos módulos de OMNeT++ de tipo *Car* en los instantes 0 y 8 (usando las imágenes *car0_vs* y *car1_vs* respectivamente). Más adelante en esta memoria, se prestará atención especial a este tipo de módulo de OMNeT++. De igual forma, el simulador de tráfico rodado puede eliminar del simulador de red todos los vehículos que ya hayan alcanzado su punto de destino. Para ello, hace uso del comando *del*.

Las trazas de movilidad se comunican en forma de listado de comandos tipo *mov* con información de las velocidades y posiciones actuales de todos los nodos que hayan sido instanciados. La posición viene expresada tanto en el formato de coordenadas de OMNeT++, como con el indicativo de segmentos de la red que utiliza SUMO.

4.4.3 Impacto del IVC en la movilidad

A continuación se pretende hacer una presentación basada en [Som08] de las mencionadas ventajas que aporta la simulación bidireccional acoplada, mediante uso de VEINS. Para ello, se van a evaluar dos protocolos IVC: el primero consiste en un *Sistema de Información de Tráfico* (TIS) centralizado que hace uso de una infraestructura previamente instalada en la red vial y un protocolo MANET estándar sobre TCP; mientras que el segundo consiste en un TIS descentralizado sin tal infraestructura y basado en *broadcasting* de datagramas UDP. Por simplicidad, el ICV se analizará para un escenario de tipo *grid* (o de cuadrícula).

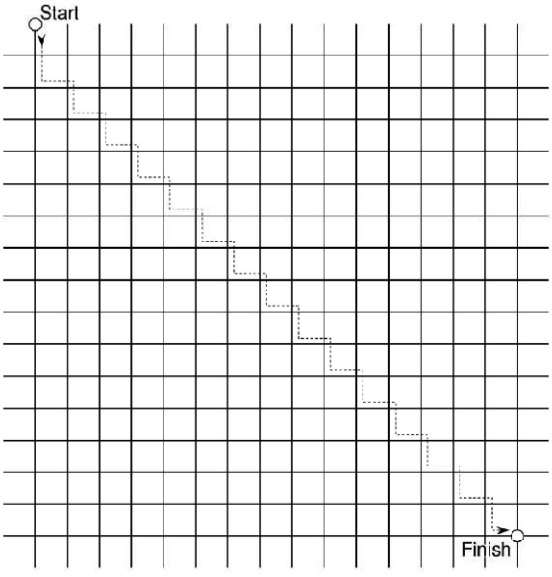


Figura 25. Escenario tipo grid simulado, donde se indican los puntos de inicio y final del recorrido de los vehículos

Esta red de vías de circulación está compuesta por segmentos de carril único, por lo que, cuando el simulador introduzca obstrucciones (deteniendo vehículos durante un tiempo), unos nodos no podrán adelantar a otros, sino que deberán encontrar un camino alternativo para sortear al obstáculo, mediante uso del IVC.

1. TIS centralizado basado en TCP y protocolos MANET estándar

En este escenario, el sistema de información de tráfico está organizado de un modo centralizado. Consta de dispositivos (*RSUs*) de monitorización de tráfico en las carreteras, y los coches trabajan como sensores. Los datos obtenidos por el sistema son transferidos a un servidor central en el *Centro de Información de Tráfico* (o TIC), donde se analiza la situación actual de la red viaria, para posteriormente ser transmitida a todos los coches presentes, mediante *broadcast*. Por otro lado, el enrutamiento vehicular se lleva a cabo mediante un protocolo MANET estándar (en concreto DYMO [Sdd08]). Podemos ver el escenario concreto en la Figura 26 a continuación, donde se muestra la reacción ante un incidente.

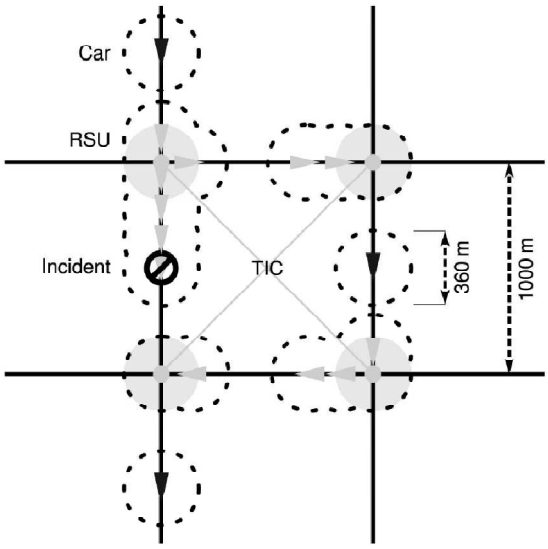


Figura 26. Escenario IVC sobre TCP. La comunicación es soportada por RSUs [Som08]

2. TIS distribuido basado en comunicación broadcast sobre UDP

Se trata de una aproximación completamente diferente a la anterior. Cada vehículo informa a los demás sobre el estado del tráfico, mediante IVC, por lo que el análisis de la situación se realiza localmente en cada coche. En la Figura 27 que vemos a continuación,

se puede observar que la reacción de los vehículos ante un incidente es totalmente distinta a la del caso anterior. Al recibir un aviso de incidencia, el vehículo trata de evitar el carril en cuestión.

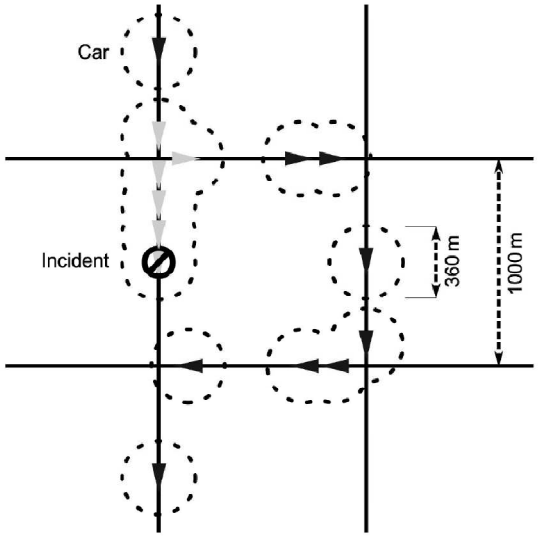


Figura 27. Escenario IVC sobre UDP. La comunicación es descentralizada [Som08]

Con la intención de examinar el impacto de las diferentes modalidades de IVC, se mide la velocidad media efectiva de los vehículos simulados (para una simulación en pequeña y en gran escala). Se puede ver el resultado en la Figura 28 que se muestra a continuación.

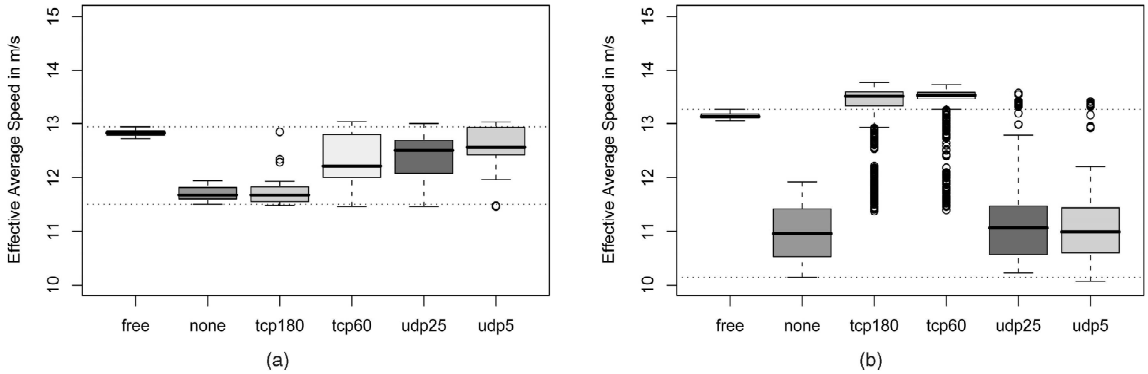


Figura 28. Velocidad media para todos los vehículos. Escenario con tráfico fluido sin obstrucciones, otro escenario sin IVC y cuatro con comunicaciones VANET simuladas en VEINS (tiempos entre mensajes de 180, 60, 25 y 5 segundos). (a) Grid de 5x5. (b) Grid de 16x16

Como se puede ver, para el caso del tráfico fluido sin incidencias, la distribución de la velocidad media para ambos escenarios es prácticamente idéntica, pero nunca se alcanzan velocidades altas. Esto es debido al hecho de que los coches siempre deceleran en las intersecciones, combinado con la alta densidad de tráfico en las vías (hay atascos porque todos los vehículos toman la ruta más corta). En el segundo caso, cuando sí hay obstrucciones, pero no IVC, obviamente la velocidad media es menor, puesto que a los efectos del caso anterior, se suma el hecho de que haya interrupciones en el tráfico.

Por el contrario, cuando se introduce el protocolo de aviso de incidentes, se puede observar el impacto del IVC. Vemos que dependiendo de la escala de la simulación, los distintos escenarios para el IVC se comportan de diferente forma. Para una simulación de pequeña escala (Figura 28 *a*), el protocolo sobre TCP con transmisión de mensajes cada 180 segundos no tiene un efecto importante en el rendimiento del sistema (demasiado tiempo entre mensajes), mientras que para el caso de utilizar la comunicación sobre UDP, se consiguen resultados bastante mejores. Vemos, por tanto, como el IVC logra que los vehículos no sufran un incremento del tiempo de viaje por culpa de los incidentes simulados.

Por otra parte, para una simulación a gran escala (Figura 28 *b*), los resultados se puede decir que son justo los contrarios. Para el caso de la comunicación sobre TCP, con tiempos entre mensajes altos, el comportamiento es muy bueno. Se consigue que casi todos los vehículos lleguen a su destino incluso más rápido que para el caso en que no hay obstrucciones (y no hay IVC).

4.5 Resumen

En el presente capítulo se han presentado las herramientas de código abierto sobre las que se sustenta el marco de simulación VEINS: el simulador de tráfico SUMO, el simulador de red OMNeT++ y el módulo de comunicación TraCI.

Se ha visto el principio de funcionamiento de cada simulador y sus prestaciones fundamentales. Más adelante, concretamente en el capítulo 6, se explicará en detalle su uso práctico.

A parte de dar a conocer importantes simuladores para el ámbito de las VANETs, como son SUMO y OMNeT++, se han presentado ideas muy significativas que deben tenerse presentes para el estudio de su simulación. Concretamente, se ha visto que un acoplamiento bidireccional entre el simulador de red y el de tráfico, nos ofrece grandes ventajas sobre otras soluciones desacopladas u *offlines*.

El marco de simulación VEINS nos brinda toda la funcionalidad necesaria para llevar a cabo esta simulación bidireccionalmente acoplada, como ha comprobado mediante ejemplos.

Capítulo 5

Instalación y puesta en marcha

5.1 Introducción

El presente capítulo tendrá como fin presentar al lector una guía detallada de instalación y puesta en marcha del simulador VEINS. Esto implicará la instalación de cada de uno de los módulos que lo componen, así como elementos accesorios, pero necesarios para el correcto funcionamiento del sistema y, por tanto, de las simulaciones.

Para hacer de esta una guía lo más completa posible, se presenta a continuación la serie de pasos necesarios para la instalación y puesta en funcionamiento del sistema completo, tanto bajo plataforma Windows, como Linux.

Cabe indicar, a modo de aclaración, que en la guía que se muestra a continuación se establecerán puntos diferenciados en caso de que el procedimiento a seguir sea distinto para Linux y para Windows. De no indicarse nada, el lector debe asumir que los pasos que se exponen, los cuales debe llevar a cabo para la correcta instalación, son totalmente idénticos para los dos sistemas.

5.2 Instalación

A continuación se muestran, por orden, los pasos necesarios para que el sistema completo de simulación VEINS pueda funcionar correctamente.

Se asume que el usuario dispone del sistema operativo Windows 7 (32-bit) (aunque la gran mayoría de las pautas aquí indicadas serán válidas también para otras versiones) o alguna distribución de Linux, pero no cuenta con ninguno de los elementos que requiere VEINS. Por este motivo, se muestran todos y cada uno de los pasos a seguir, pese a que para ciertos usuarios puede que alguno no sea necesario.

5.2.1 Intérprete de Python

Como es sabido, Python es un lenguaje de programación de alto nivel, caracterizado por ser multiparadigma (esto significa que soporta tanto orientación a objetos, programación imperativa, como programación funcional) y multiplataforma. Fue creado a finales de los años 80 en el *National Research Group for Mathematics and Computer Science*, en Holanda.

Cuenta con una licencia de código abierto y es por ese motivo, entre otros, por el que SUMO permite desarrollar *scripts* en este lenguaje. Por lo tanto, para el funcionamiento de nuestro simulador de tráfico, se necesitará tener un intérprete de Python instalado.

Son muchos los intérpretes que podemos encontrar a nuestra disposición. En este caso, se propone instalar *Active Python Community Edition*, por tratarse de un intérprete gratuito y que ofrece buenas prestaciones.

En la web de *Active State* [ASL], en la sección *Productos* podemos encontrar *Active Python Community Edition*. Sólo es preciso buscar el enlace a la descarga que corresponda al instalable de la versión de Windows que se tenga instalada, o bien al archivo de Linux.

Figura 29. Intérprete Active Python Community Edition [ASL]

Como se verá más adelante (en el capítulo 6), VEINS hace uso de un *script* de Python que es de vital importancia para el correcto desarrollo de las simulaciones.

5.2.2 Intérprete de Perl

Al igual que Python, Perl es un lenguaje de programación necesario en algunos de los scripts de los que hace uso el simulador SUMO. El lenguaje de programación Perl, que fue diseñado por Larry Wall en 1987, toma características del lenguaje de programación C, del lenguaje interpretado Shell, AWK, Sed, Lisp y, en un grado inferior, de muchos otros lenguajes. Estructuralmente, Perl está basado en un estilo de bloques como los del C o AWK, y fue ampliamente adoptado por su destreza en el procesado de texto y por no tener ninguna de las limitaciones de los otros lenguajes de script.

Se recomienda al usuario acceder directamente al sitio web de Perl [PPL] donde tendrá acceso a la última versión para su sistema Linux. En caso de utilizar Windows, la implementación de código abierto más extendida corresponde a *Strawberry Perl* [SBP].

5.2.3 Microsoft Visual Studio

A partir de la versión 0.9.7 de SUMO, este simulador ha sido desarrollado utilizando *Microsoft Visual Studio 2005 SP1*. Es por este motivo, por el que en ciertas configuraciones, si se está utilizando plataforma Windows y se experimentan problemas a la hora de arrancar SUMO, es recomendable instalar este paquete.

Directamente desde las librerías online de Microsoft, se puede acceder a esta herramienta. Concretamente en la URL que se indica en [MSDN].

5.2.4 Entorno de ejecución Java (JRE)

Finalmente, para el correcto funcionamiento de OMNeT++, el usuario debe instalar en su sistema el entorno de ejecución Java (JRE). El entorno de desarrollo integrado de simulación de OMNeT++, basado en Eclipse, requiere de este lenguaje.

Directamente desde el sitio web de Java (ver [Java]), el usuario tiene acceso a la última versión tanto para el instalable de Windows, como paquetes o binarios autoextraíbles para Linux.

5.2.5 Compilador de C++

En caso de utilizar Windows, no es necesario ningún otro software, a parte de los ya mencionados, pues en el paquete de instalación de OMNeT++ se incluyen otros

auxiliares, tales como un compilador de C++ (de vital importancia para este simulador de red). No obstante, para determinadas versiones de Linux, el usuario deberá acudir al repositorio de *software* de su distribución para descargar *gcc*.

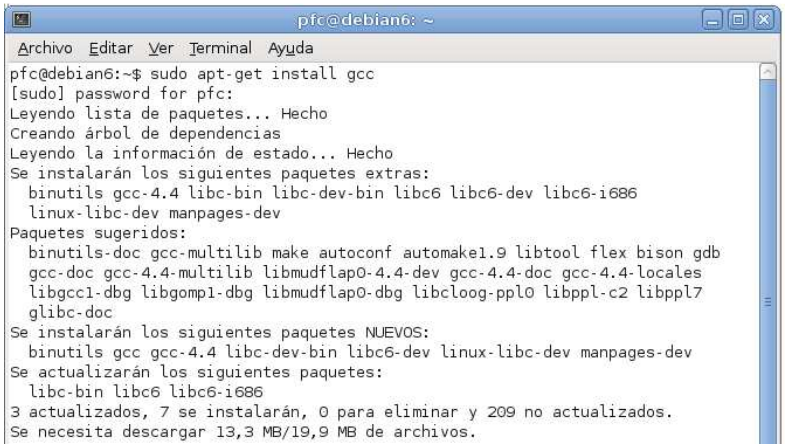


Figura 30. Instalación del compilador de C++ gcc en Linux

5.2.6 Instalación de SUMO

Una vez que se dispone del software auxiliar anterior funcionando, el sistema operativo está listo para que instalemos y ejecutemos el simulador de tráfico SUMO.

En la página de SUMO en *SourceForge* [SDown], se cuenta con una wiki con consejos para su instalación, así como con los paquetes para su descarga. Puesto que el programa se encuentra en estado continuo de desarrollo, se recomienda prestar atención para descargar la versión que sea de nuestro interés. Como se verá a continuación, para el desarrollo de esta memoria no se ha escogido la última versión de SUMO, por motivo de encontrar incompatibilidades con OMNeT++, sobre esto se hablará más en la sección de *Puesta en marcha y depuración*. En concreto, todo lo que se expondrá aquí ha sido realizado con la versión 0.12.1.

Las versiones están a disposición del usuario a través de la herramienta *Subversion*. Existe un gran número de clientes *Subversion* para todas las plataformas. No obstante, se recomienda el uso de algún cliente de línea de comando (por ejemplo, Git-SVN), ya que no se requiere de ninguna funcionalidad muy avanzada.

A través del siguiente comando accedemos a la versión de SUMO que deseemos, mediante búsqueda en el directorio:

$ svn co https://sumo.svn.sourceforge.net/svnroot/sumo/trunk/sumo

- **Particularidades en Windows**

En el caso de Windows, simplemente se descargará y ejecutará el paquete precompilado que se ofrece, pues todo lo necesario para el funcionamiento de SUMO está ya incluido. Se debe descargar el archivo llamado *sumo-winbin-<versión>.zip* y descomprimirlo el carpeta que se desee usando Winrar, Winzip, o una herramienta similar. En el destino elegido, se podrá encontrar una carpeta llamada *bin*, es en ella donde están los ficheros ejecutables y, concretamente, *sumo-gui.exe* (en Windows 7 se deberá ejecutar como administrador, mediante clic en el botón derecho).

- **Particularidades en Linux**

Por otra parte, en caso de utilizar Linux, tras la bajada del código fuente (ejecución del comando mostrado anteriormente), se necesita ejecutar *Makefile.cvs*, pues se requiere una llamada a las herramientas *Autotools* (*GNU Build System*), de lo que se encargará este fichero. El comando, por tanto, sería:

$ make -f Makefile.cvs

Finalmente, para construir los binarios de SUMO, es necesario llamar al *script configure*. Existen opciones tales como instalación el directorio raíz, etc. que pueden consultarse mediante llamada a la opción *--help*. La ejecución, básicamente, es de la siguiente forma:

$ ./configure [opciones]
$ make

5.2.7 Instalación de OMNeT++

El siguiente paso será instalar el sistema de simulación de red OMNeT++. En primer lugar, se debe descargar la versión deseada de la sección de descargas en el sitio web de OMNeT++ indicado en [OMNeT]. Para el desarrollo de este trabajo, la versión escogida es la 4.0.

Veamos primero las particularidades para cada sistema operativo y, seguidamente, los pasos comunes.

- **Particularidades en Windows**

En el caso de estar trabajando en Windows, el paquete descargado es prácticamente autónomo: además de los ficheros de OMNeT++, incluye un compilador de C++, un entorno de desarrollo en línea de comandos y todas las librerías y programas requeridos por OMNeT++. Para realizar la llamada a los comandos mencionados, el usuario debe ejecutar *mingwenv.cmd*, de forma que aparecerá una consola con la *shell* de *MSYS*.

- **Particularidades en Linux**

En el caso de estar usando plataforma Linux, se deben tener en cuenta algunos puntos. En primer lugar, el simulador puede instalarse en las siguientes distribuciones:

• *Ubuntu 10.04 LTS, 11.04*
• *Fedora Core 15, 16*
• *Red Hat Enterprise Linux Desktop Workstation 5.5*
• *OpenSUSE 11.4*

Una vez descargada la versión que se desee, se deberá llamar al siguiente comando en la carpeta donde se desee instalar OMNeT++:

$ tar xvfz omnetpp-4.0-src.tgz

OMNeT++ necesita que su directorio *bin/* se encuentre en el *path*, para ello se dispone de un *script* que realizará esta labor. Se debe hacer la siguiente llamada (se puede ver una captura de pantalla en la Figura 34 mostrada a continuación):

$ cd omnetpp-4.0/
$. setenv

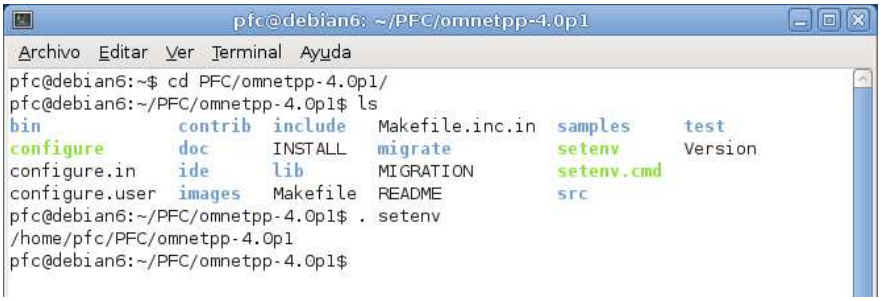


Figura 31. Carpeta de instalación de OMNeT++ y llamada al script setenv en Linux

Aclaradas las particularidades para cada sistema, se exponen a continuación los puntos comunes.

La carpeta que se seleccione como destino de la instalación debe ser tal que la ruta no contenga ningún espacio. Por ejemplo, OMNeT++ no se debe ubicar en la carpeta de *Archivos de Programa*. En el directorio creado podremos encontrar carpetas llamadas *doc*, *images*, *include*, *msys* y *etc*, además de los ficheros *mingwenv.cmd*, *configure*, *Makefile*, entre otros. Para comenzar el proceso de construcción se debe llamar a los siguientes comandos. Así se realizará la configuración mediante llamada al *scritp configure*, el cual comprueba el software instalado, para escribir el resultado en el *makefile*. Finalmente se ejecutará este fichero para realizar la compilación:

$ ./configure
$ make

Como se puede ver en la Figura 31 y en la Figura 32 siguientes, de esta forma se realizan las comprobaciones y configuraciones necesarias para que OMNeT++ esté listo para ser usado. Se debe prestar atención a los posibles *warnings* que se muestran, aunque si se han seguido los pasos aquí explicados y se ha comprobado que se tiene instalado lo que se ha indicado, no habrá problemas.

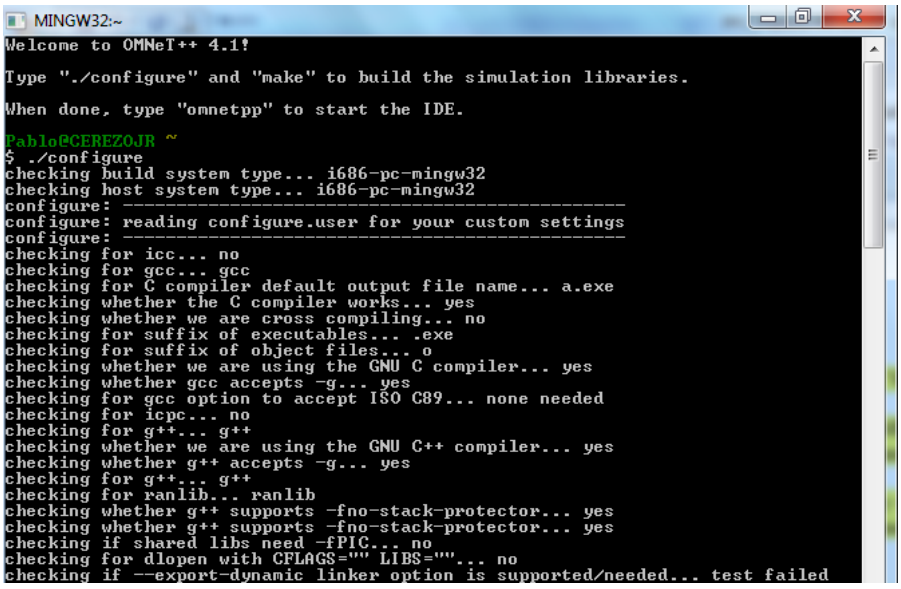


Figura 32. Llamada a ./configure para la instalación de OMNeT++

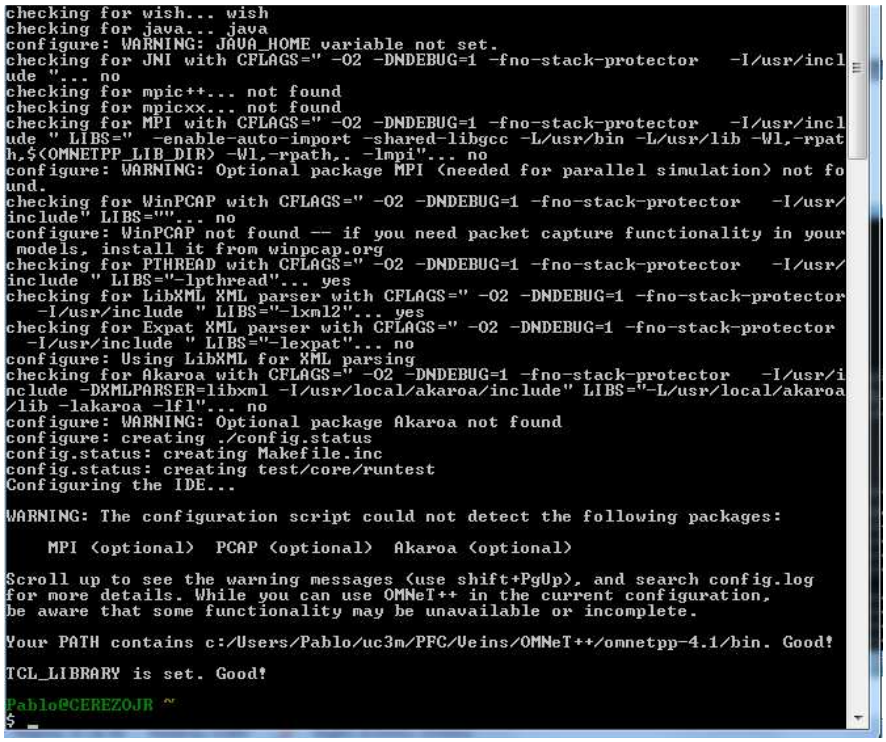


Figura 33. La instalación de OMNeT++ ha finalizado con éxito

Como ya se ha mencionado, y se verá más en detalle en el siguiente capítulo, este simulador cuenta con entorno de desarrollo integrado (*IDE*) basado en Eclipse. Una vez finalizados los pasos anteriores, el usuario puede ejecutarlo mediante el comando:

$ omnetpp

De esta forma, se cargará OMNeT++ listo para que podamos trabajar con él.

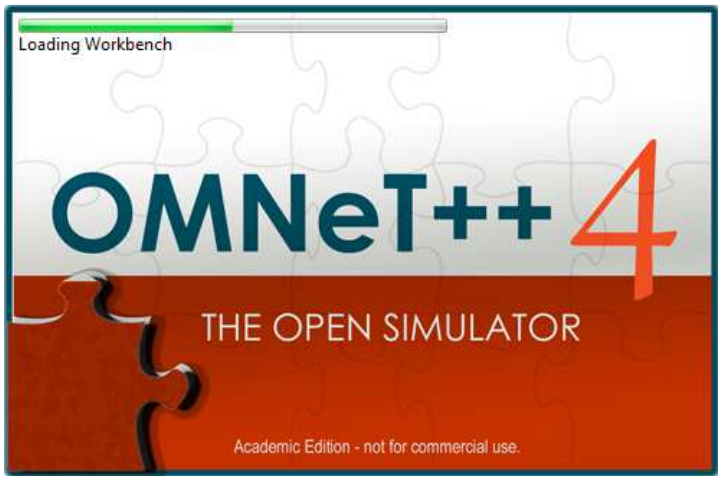


Figura 34. Una vez finalizada la instalación: carga del IDE de OMNeT++

5.2.8 Módulos de cliente TraCI

Una vez instalados SUMO, OMNeT++ y el software auxiliar ya mencionado, lo último que necesitamos para poder realizar simulaciones con VEINS es el paquete TraCI.

Este proceso se realizará desde el propio entorno de desarrollo de OMNeT++. Para abrirlo, como ya se ha indicado, se ejecutará *omnetpp* en la consola. El siguiente paso será hacer clic en *File* > *Import…* > *Git* > *Git Repository* e introducir la dirección:

git://github.com/sommer/inet-sommer.git

Se puede ver el diálogo de importación en la Figura 35 mostrada a continuación. Finalmente, se debe construir el proyecto generado mediante clic en el menú *Project* > *Build All*. Tras este proceso, el cual puede llevar un tiempo considerable, el usuario estará en disposición de lanzar las primeras simulaciones para testear el IVC de su interés.

Figura 35. Importación del módulo TraCI desde OMNeT++

5.3 Puesta en marcha y depuración

Este apartado tiene como finalidad facilitar al lector la puesta en marcha del sistema VEINS tras su instalación (realizada mediante los pasos descritos anteriormente).

De nuevo, cabe destacar que todo el trabajo realizado, previo a la redacción de esta memoria, ha tenido lugar utilizando las siguientes versiones de los simuladores SUMO y OMNeT++:

SUMO: *versión 0.12.1*.
OMNeT++: *versión 4.0.*

Se desea hacer hincapié en este punto, pues por problemas de incompatibilidades entre ciertos módulos, que no se han visto descritos en ningún lugar de la literatura de VEINS, las simulaciones no se producían correctamente para la combinación de otras versiones diferentes a estas. Esto ocasionó una inversión temporal considerable, tanto en reconocer la existencia del problema, como en encontrar la pareja que sí ofreciera un comportamiento más estable.

Por lo tanto, se recomienda al usuario comenzar a trabajar con estas versiones. No obstante, el rápido desarrollo de estas herramientas de código abierto hace que surjan cambios de forma relativamente rápida, por lo que posibles *bugs* que dieran lugar a errores en determinadas versiones, pueden haber sido reparados para otras más modernas.

Veamos, a continuación, los pasos para comprobar que VEINS está listo para su uso. Los pasos que se muestran a continuación son válidos trabajando bajo ambas plataformas: Windows y Linux.

Comprobación del correcto funcionamiento de SUMO

En la ventana de línea de comandos de OMNeT++ (*MinGW*) el usuario debe ser capaz de simular un escenario de ejemplo de SUMO. Para ello, se debe ir directorio donde esté instalado OMNeT++ y buscar la siguiente ruta:

… /omnetpp-4.0/samples/inet-sommer/examples/traci-launchd/

Desde él se debe realizar la prueba de ejecución de la simulación de ejemplo de SUMO, mediante la siguiente llamada a *sumo-gui* (en Linux la llamada será *sumo-gui*, mientras que en Windows será *sumo-gui.exe*). Se muestra el resultado de esta llamada en la Figura 36 a continuación (los puntos suspensivos deben ser sustituidos por el directorio de instalación de SUMO):

… /sumo-0.12.1/bin/sumo-gui.exe -c sumo.sumo.cfg

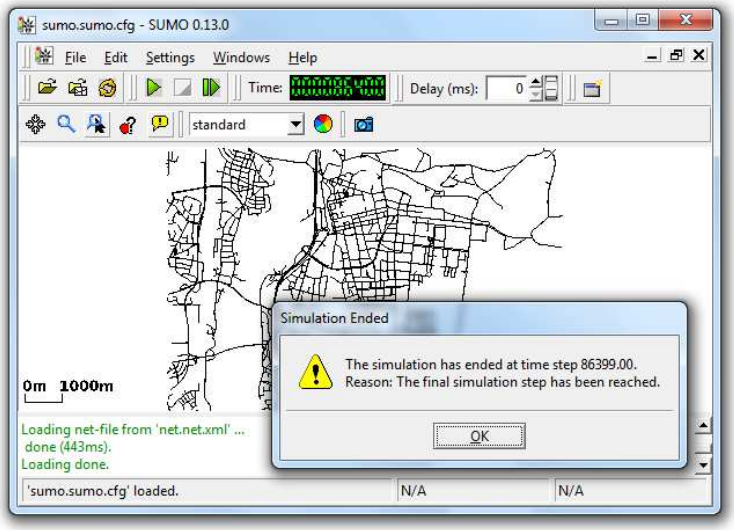


Figura 36. Resultado de la simulación de ejemplo de SUMO

Comprobación del correcto funcionamiento de OMNeT++

El usuario debe ser capaz de simular un escenario de ejemplo en el entorno de desarrollo de OMNeT++. Para ello, se hará uso del proyecto importado anteriormente.

En el árbol de directorios, desde la raíz donde se ha instalado OMNeT++, se debe buscar la siguiente ruta:

… /inet-sommer/examples/ethernet/lans/

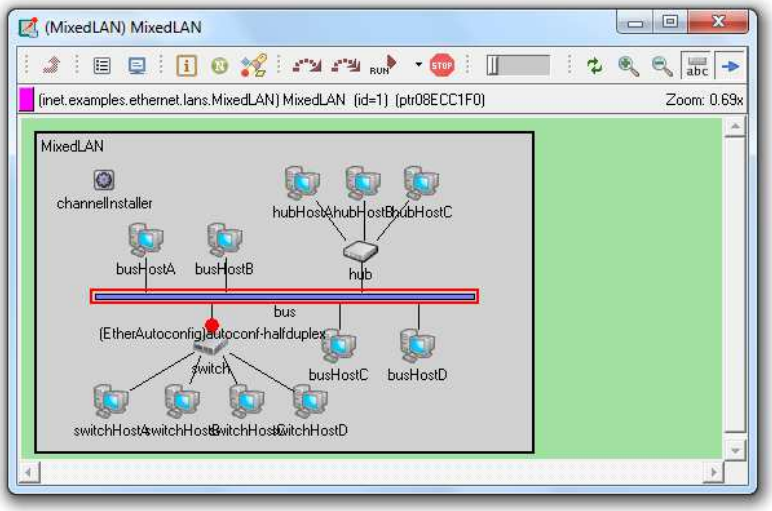


Figura 37. Ejecución de la simulación de ejemplo de OMNeT++

Mediante clic con el botón derecho en el fichero *omnetpp.ini* y elección de la opción *Run As > OMNeT++ simulation*, el usuario debe ver como se crea la simulación de una LAN. Seleccionando una configuración de red y haciendo clic en el botón de *Run*, la simulación comenzará a ejecutarse, tal y como se observa en la Figura 37.

5.4 Resumen

En este capítulo se han mostrado los pasos necesarios para que el usuario tenga su sistema listo para realizar simulaciones con VEINS. Se ha indicado cómo instalar cada uno de los componentes básicos, así como algún software auxiliar, todo ello, tanto para una plataforma Windows, como Linux.

Posteriormente, se ha hecho una recomendación en cuanto a las versiones a utilizar y se han dado unos consejos de utilidad para que el usuario tenga certeza de que su sistema está preparado para lanzar sus simulaciones con VEINS.

En el siguiente capítulo se verá en profundidad cómo realizar estas simulaciones: qué funcionalidades concretas nos ofrece OMNeT++, así como SUMO. Finalmente, en el apartado 6.4, se mostrará cómo ambos funcionan en paralelo, gracias a TraCI.

Capítulo 6

Uso del simulador

6.1 Introducción

Una vez expuesto el problema y la necesidad de la simulación, así como las claras ventajas que la bidireccionalidad entre simuladores nos aporta, se ha presentado el conjunto de herramientas de las que disponemos y sobre las que se sustenta VEINS.

Es ahora el momento de mostrar en profundidad los pasos prácticos necesarios para desarrollar un escenario de simulación. Este capítulo, por tanto, tiene dos bloques claramente diferenciados. En primer lugar, se estudia la creación de proyectos en OMNeT++ y, tras ello, la construcción de redes de vehículos en SUMO. Para cada uno de ellos se estudiarán todas las posibilidades que se ofrecen al usuario de forma general y con casos prácticos, siempre enfocándonos en el campo de nuestro interés: las VANETs.

Para finalizar, se presentará el uso conjunto de ambos simuladores en VEINS. Se verá qué parámetros de medida de la simulación nos devuelve el sistema y cómo conseguirlo. Una serie de ejemplos prácticos con diversos escenarios típicos o sobre los que trabajar en aplicaciones más complejas, permitirán clarificar todo lo explicado.

6.2 Proyectos OMNeT++

6.2.1 Visión general

Se han descrito en profundidad los principios de funcionamiento del simulador de redes OMNeT++, sus características a alto nivel, las ventajas sobre otros simuladores y su papel en VEINS. Asimismo, se han detallado los pasos necesarios para tener el sistema completo y listo para trabajar con él. Con todo ello presente, en esta sección se ofrece información sobre el trabajo con OMNeT++ en la práctica. Se van a discutir temas tales como ficheros necesarios para realizar los modelos y cómo compilar y ejecutar las simulaciones.

Un modelo OMNeT++ se compone de las siguientes partes [OMNeT]:

- Descripción de la topología mediante lenguaje NED. Estos archivos describen la estructura de cada módulo: parámetros, puertas, etc. Los archivos NED pueden ser escritos mediante cualquier editor de texto. No obstante, OMNeT++ IDE ofrece un excelente soporte como interfaz gráfica y para edición de texto.

- Definición de mensajes (archivos *.msg*). Se pueden definir varios tipos de mensajes diferentes y agregar campos de datos a los mismos. OMNeT++ traducirá las definiciones de mensajes a clases C++ completamente desarrolladas.

- Fuentes de módulos simples. Se trata de archivos C++ con extensión *.h* /*.cc*.

El sistema de simulación proporciona los siguientes componentes:

- *Kernel* de la simulación. Como ya se mencionó en el capítulo 4, éste contiene el código que maneja la simulación y la librería de la clase de simulación. Está escrito en C++ y compilado en una librería estática o compartida.

- Interfaces de usuario. Las interfaces de usuario en OMNeT++ se utilizan durante la ejecución de la simulación, con el fin de facilitar la depuración,

realizar una demostración, o la ejecución por lotes de las simulaciones. Están escritas en C++ y compiladas en librerías.

Los programas de simulación se desarrollan a partir de todos estos componentes. En primer lugar, los ficheros *.msg* son traducidos a código C++ usando el programa *opp_msgc*. Tras ello, todos los ficheros fuente C++ son compilados y vinculados con el *kernel* de simulación y una librería de interfaz de usuario para crear un ejecutable de simulación o una librería compartida. Los ficheros NED se cargan dinámicamente en su forma original como texto cuando el programa de simulación comienza.

En lo que se refiere a la ejecución de las simulaciones y el análisis de los resultados obtenidos, cabe mencionar que una simulación puede ser compilada como un programa ejecutable independiente, por lo que puede ser ejecutada en otras máquinas (sin necesidad de que OMNeT++ esté presente) o puede ser creada como una librería compartida.

Cuando un programa se inicia, en primer lugar lee todos los ficheros NED (los cuales contienen, como ya se ha indicado, la topología del modelo). A continuación, se lee el fichero de configuración (normalmente llamado *omnetpp.ini*). Este fichero contiene los valores que controlan cómo se ejecuta la simulación, valores para los parámetros del modelo, etc.

Cabe hacer una mención de la interfaz de usuario que proporciona OMNeT++. Su propósito es hacer visible para el usuario las características internas del modelo, permitirle controlar la ejecución de la simulación, así como darle la posibilidad de intervenir modificando variables u objetos del modelo. Todo esto es de enorme utilidad en la etapa de desarrollo y *debugging*.

6.2.2 Lenguaje NED

NED, que son las siglas de *NEtwork Description* (*Descripción de la Red*), es el lenguaje en el que el usuario de OMNeT++ describe la estructura de su modelo de simulación. NED permite al usuario declarar los módulos simples de su simulación, conectarlos y unirlos en módulos compuestos.

Este lenguaje tiene las siguientes características principales, las cuales le otorgan una muy buena escalabilidad para proyectos grandes:

- **Jerárquico**

Cada módulo que pueda ser demasiado complejo, puede descomponerse en módulos más pequeños, que a su vez se podrán unir en un módulo compuesto.

- **Herencia**

Los módulos y canales pueden implementarse mediante herencia. Los módulos o canales derivados (es decir, los hijos) pueden añadir nuevos parámetros, puertas y (en el caso de módulos compuestos) nuevos submódulos y conexiones entre ellos. Pueden dar valores fijos a parámetros ya existentes.

- **Basado en componentes**

Los módulos simples, así como los compuestos, son reutilizables mediante el mecanismo de herencia. Esto permite que existan librerías de componentes (como es el caso del marco INET o MiXiM).

- **Paquetes**

El lenguaje NED ofrece una estructura de paquetes similar a la de Java, para reducir el riesgo de conflictos de nombres entre diferentes modelos. Se introduce NEDPATH (similar al CLASSPATH de Java), con el objeto de hacer más fáciles las dependencias entre modelos de simulación.

- **Tipos internos**

Los tipos de canales y módulos que se vayan a usar localmente en un módulo compuesto, pueden ser definidos como internos, de modo que se reduzcan los problemas con el nombrado de los mismos.

- **Metadatos**

Existe la posibilidad de añadir metadatos a los módulos, canales, parámetros, puertas o submódulos. Estos metadatos no son usados directamente por el *kernel* de simulación, pero pueden ser útiles como información adicional. Un metadato, por ejemplo, puede ser una representación gráfica del módulo (un icono).

Tras esta presentación de las características de NED (que se entenderán mejor con los casos prácticos que se van a ver a continuación), en el siguiente apartado se presentarán algunos componentes o funcionalidades extras de este lenguaje, que son requeridas para modelar la red móvil de nuestro interés, la red vehicular. Exponer cada una de las posibilidades que nos brinda OMNeT++ supera los objetivos de este proyecto. No obstante, el lector puede encontrar en [OMNManual] una guía muy detallada y extensa con todas las variantes y opciones que nos ofrece este simulador y este lenguaje.

6.2.3 Módulos OMNeT++ de VEINS

Una vez hecha la introducción sobre lo que necesita OMNeT++ para realizar una simulación, veamos cómo debe ser nuestro proyecto para realizar la simulación de la red vehicular en concreto.

Tanto los ficheros de descripción de la topología (*.ned*), como el de iniciación (*.ini*) que se van a exponer y analizar aquí, han sido desarrollados dentro del marco de simulación VEINS, por lo que el usuario no tendrá que implementarlos. Bastará con seguir los pasos descritos en el capítulo anterior, para tener a nuestra disposición todos los ficheros necesarios.

Por lo tanto, VEINS nos ofrece el armazón de OMNeT++ sobre el que trabajar. Eso sí, será necesario que el usuario modifique algunos parámetros o incluso módulos, para que la simulación haga lo que se desea (protocolo de transporte a implementar o la capa de aplicación, entre otros). Sobre esto se ahondará a continuación.

Dentro del proyecto OMNeT++ que se ha creado mediante la importación del módulo TraCI (ver apartado 5.2.1.7 de esta memoria), podemos ver los siguientes elementos, necesarios para nuestra simulación:

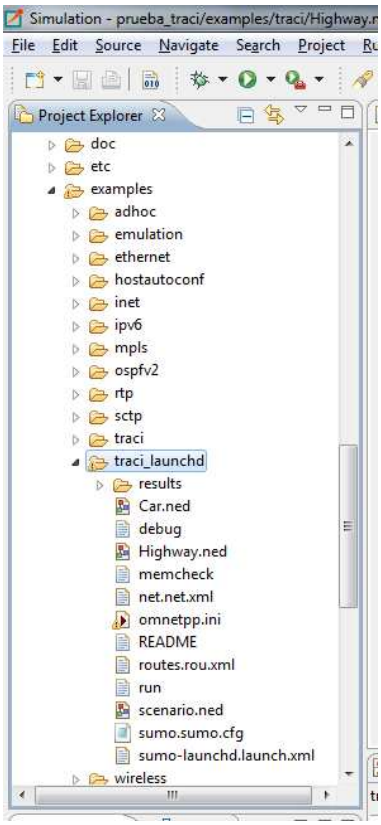


Figura 38. Árbol de ficheros dentro del proyecto creado en OMNeT++

Como se puede ver, para la utilización de VEINS no es precisa la creación de un proyecto OMNeT++ nuevo, pues mediante el proceso de importación contemplado en la instalación de los componentes de VEINS, se ponen a disposición del usuario todos los módulos necesarios. No se entrará, por tanto, en la descripción de cómo crear un proyecto desde cero en OMNeT++, ya que realmente escapa del ámbito del uso de VEINS. No obstante, en caso de interés o necesidad para otras aplicaciones no relacionadas este marco de simulación, se remite al usuario a [OMNManual].

A continuación se analizan al detalle todos los módulos que componen este proyecto.

6.2.3.1 Escenario

scenario.ned

El módulo sobre el que se construye la red es *scenario.ned*. Fija los pilares básicos de la topología de la red y en este caso va, simplemente, a heredar de *Highway.ned*, que será quien defina qué submódulos formarán la simulación. Es una forma de trabajar siempre con el mismo fichero de escenario, teniendo el usuario que modificar únicamente el fichero del que éste hereda (en función de lo que se desee simular).

```
network scenario extends Highway
{
}
```

Highway.ned

Como se ha dicho, es este fichero *.ned* el que realmente define la topología básica (pues el denominado *scenario* hereda de él).

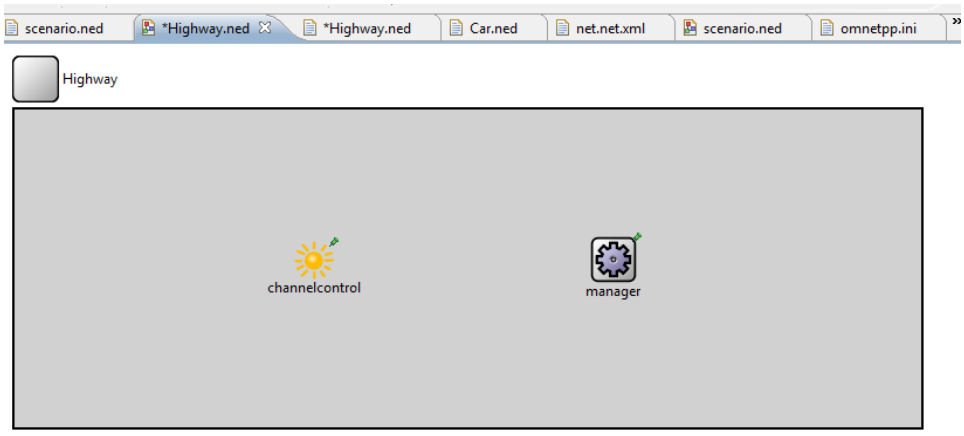


Figura 39. Visualización gráfica de los submódulos que componen al módulo Highway.ned

En la Figura 39 mostrada previamente, podemos ver la representación gráfica de los submódulos que componen el módulo *Highway*.

Este módulo debe estar compuesto por los dos que se observan en la figura:

- Módulo que controle (simule) el comportamiento del canal: *channelcontrol*.
- Módulo que gestione la movilidad de los nodos (vehículos): *manager*.

Veamos y analicemos el código que hay detrás de esta red:

```
// Copyright (C) 2008 Christoph Sommer <christoph.sommer@informatik.uni-erlangen.de>
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.

package inet.examples.traci;

import inet.world.ChannelControl;
import inet.world.traci.TraCIScenarioManager;

module Highway
{
    parameters:
        double playgroundSizeX;
        double playgroundSizeY;
    submodules:
        channelcontrol: ChannelControl {
            parameters:
                playgroundSizeX = playgroundSizeX;
                playgroundSizeY = playgroundSizeY;
                @display("p=256,128");
        }
        manager: TraCIScenarioManager {
            parameters:
                @display("p=512,128");
        }
    connections allowunconnected:
}
```

Previo al análisis, cabe notar que en la cabecera de este código podemos ver que ha sido desarrollado dentro del marco de simulación VEINS (en la Universidad de Erlangen, Alemania) y que se trata de software libre. Cualquier porción de código mostrada en esta memoria, posee idénticas características.

Como se observa, este módulo está compuesto por los dos módulos simples que ya se han anunciado. Veamos qué aporta exactamente cada uno de ellos.

En primer lugar, el submódulo de control del canal, denominado *ChannelControl*, es el que modela los efectos derivados del hecho de que la comunicación se produzca entre nodos inalámbricos y móviles. Recibe información de la localización y el movimiento de los nodos y determina cuáles se encuentran a distancia donde la comunicación es posible y cuáles donde se reciben interferencias. Esta información es utilizada por los interfaces radio de los nodos, en el momento en que se vaya a producir la transmisión.

En segundo lugar, requerimos de una entidad que sea responsable de crear nodos dinámicamente, así como de controlar que su movimiento sea el que indica la información recibida desde el simulador de tráfico. Para todo ello tenemos el denominado *manager*, que es un módulo de tipo *TraCIScenarioManager*.

Este módulo conecta OMNeT++ con el servidor TraCI encargado de ejecutar las simulaciones de tráfico rodado (es decir, SUMO). Fija y controla la simulación, moviendo los nodos con la ayuda de un módulo auxiliar denominado *TraCIMobility* (el cual está presente en cada nodo). Este otro módulo es quien recibe las actualizaciones referentes a posición y estado de los vehículos, para pasar la información a su módulo padre.

Para finalizar, la última sentencia que vemos en el código simplemente indica que no es obligatorio que todas las puertas del módulo estén interconectadas.

6.2.3.2 Nodo (vehículo)

Car.ned

Una vez presentado el módulo que controla la simulación, así como los componentes que lo conforman y sus funciones, veamos el otro módulo que juega un papel básico en el modelado de la red vehicular: el nodo (vehículo).

En la Figura 40 se puede observar la representación gráfica de este módulo, en la que se ve de manera muy ilustrativa qué funcionalidades se implementan. Como vemos, para la simulación de cada nodo son precisos una serie de componentes auxiliares (tanto para la parte telemática, como serían las tablas de enrutamiento y de interfaces, como para la parte de gestión de la movilidad), así como el modelado de cada una de las capas necesarias para que se produzca la comunicación (desde la tarjeta de red inalámbrica, hasta el nivel de aplicación, pasando por las capas de red y de transporte).

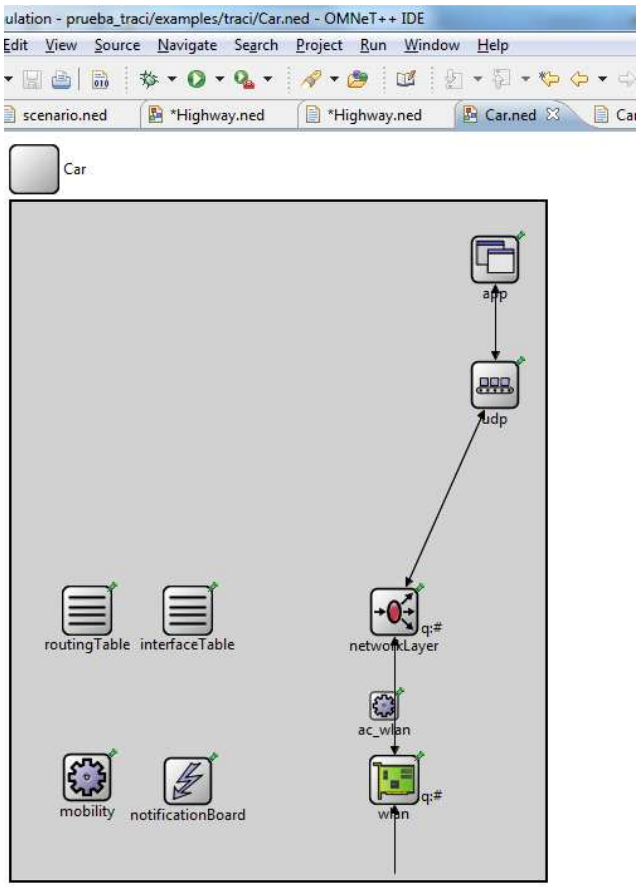


Figura 40. Visualización gráfica de los submódulos que componen Car.ned (que actúa como nodo de la red vehicular)

A continuación se presenta el código NED que hay por debajo de cada uno de los componentes que conforman el modelado del vehículo. Veamos, en concreto, qué hace cada submódulo y cómo es su interacción con los demás.

```
package inet.examples.traci;

import inet.networklayer.autorouting.HostAutoConfigurator;
import inet.transport.udp.UDP;
import inet.nodes.inet.NetworkLayer;
import inet.networklayer.ipv4.RoutingTable;
import inet.networklayer.common.InterfaceTable;
import inet.mobility.traci.TraCIMobility;
import inet.linklayer.ieee80211.Ieee80211NicAdhoc;
import inet.base.NotificationBoard;
import inet.applications.traci.TraCIDemo;

module Car
{
    parameters:

        @display("bgb=424,541");
    gates:
        input radioIn;


submodules:
```

```
        notificationBoard: NotificationBoard {
            parameters:
                @display("p=140,462;i=block/control");
        }
        ac_wlan: HostAutoConfigurator {
            @display("p=296,402");
        }
        interfaceTable: InterfaceTable {
            parameters:
                @display("p=140,326;i=block/table");
        }
        mobility: TraCIMobility {
            parameters:
                @display("p=60,459;i=block/cogwheel");
        }
        routingTable: RoutingTable {
            parameters:
                IPForward = true;
                routerId = "";
                routingFile = "";
                @display("p=60,326;i=block/table");
        }
        udp: UDP {
            parameters:
                @display("p=384,146;i=block/transport");
        }
        networkLayer: NetworkLayer {
            parameters:
                proxyARP = false;
                @display("p=304,327;i=block/fork;q=queue");
            gates:
                ifIn[1];
                ifOut[1];
        }
        wlan: Ieee80211NicAdhoc {
            parameters:
                @display("p=304,461;q=queue;i=block/ifcard");
        }
        app: TraCIDemo {
            parameters:
                @display("p=384,46;i=block/app");
        }
    connections allowunconnected:
        udp.appOut++ --> app.udp$i;
        udp.appIn++ <-- app.udp$o;

        udp.ipOut --> networkLayer.udpIn;
        udp.ipIn <-- networkLayer.udpOut;

        wlan.uppergateOut --> networkLayer.ifIn[0];
        wlan.uppergateIn <-- networkLayer.ifOut[0];

        radioIn --> wlan.radioIn;
}
```

InterfaceTable y RoutingTable

Los primeros submódulos de los considerados auxiliares son estos dos. *InterfaceTable* simplemente almacena el listado de las interfaces de red de las que dispone el nodo (el cual es actualizado por el correspondiente módulo de nivel 2). Por otra parte, el submódulo *RoutingTable* tiene como objeto almacenar la tabla de

enrutamiento de nivel 3. Esta tabla tendrá una instancia por *host* (e interfaz) en la red y podrá verse modificada durante el transcurso de la simulación, típicamente debido a la implementación del protocolo de enrutamiento (OSPF).

NotificationBoard

Este módulo permite a los nodos pasarse notificaciones unos a otros acerca de eventos que tienen lugar durante la simulación, tales como cambios en la tabla de enrutamiento, cambios en el estado de los interfaces (*up*/*down*) o en su configuración, cambios en el estado del canal inalámbrico, etc.

En definitiva, este componente actúa como intermediario entre los módulos de un nodo en cuyo estado pueden ocurrir cambios, y los interesados en conocer dichos cambios.

TraCIMobility

Este submódulo, que en *Car.ned* recibe el nombre de *mobility*, es el último de los clasificados como auxiliares. Si bien es cierto que el papel que juega es fundamental, pues el nodo en cuestión está simulando a un coche y gestionar su movilidad es de vital importancia.

Como se dijo en la explicación del módulo de escenario, en el apartado anterior, el submódulo de tipo *TraCIMobility* es controlado por *TraCIScenarioManager* y recibe de él las actualizaciones referentes a la posición física del nodo.

Ieee80211NicAdhoc

En lo que se refiere a la parte encargada del modelado de las comunicaciones entre los nodos, éste es el primer submódulo que aparece.

Ieee80211NicAdhoc (que en *Car.ned* recibe el nombre de *wlan*) está asociado con el nivel físico, así como con el de enlace y se corresponde al módulo que se introdujo en el apartado 4.3.4. Es el que implementa la interfaz de la tarjeta para las comunicaciones 802.11 y gestiona que la transferencia de información se realice de forma fiable. Podemos ver su composición exacta en la Figura 41 que se muestra a continuación.

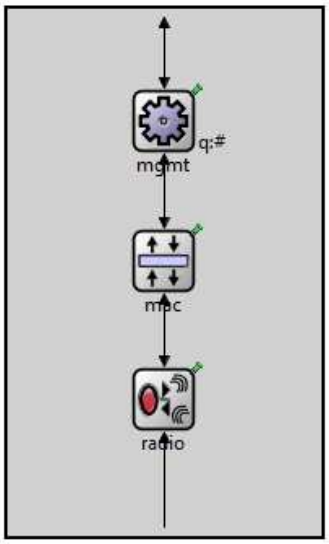


Figura 41. Submódulo wlan: Ieee80211NicAdhoc

HostAutoConfigurator y NetworkLayer

Las dos primeras capas (representadas por el submódulo anterior) se conectan directamente a la de red, como es lógico. El modelado de esta nueva capa es tarea de *NetworkLayer*. En este caso, se implementa un nodo IP, por lo que son necesarios, además, algunos submódulos dentro de éste (ICMP para la gestión de errores, IGMP y ARP). Podemos ver su composición en la Figura 42 que se muestra a continuación.

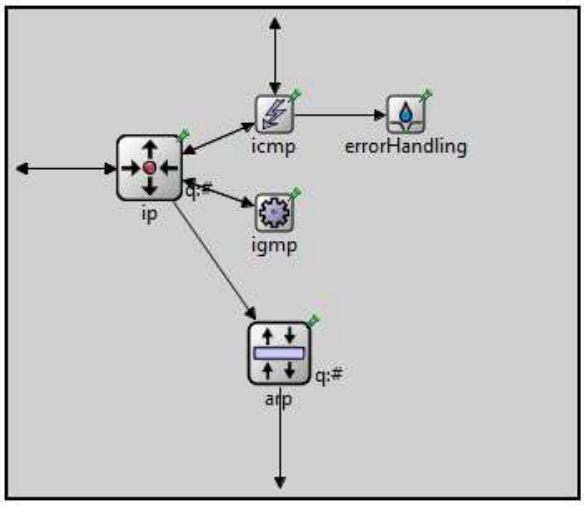


Figura 42. Submódulo networkLayer: NetworkLayer

Además, en *Car.ned* aparece un módulo más, que realiza una función auxiliar, como es la de asignar automáticamente direcciones IP y crear la tabla de enrutamiento. Para esto, se introduce *HostAutoConfigurator* (que recibe el nombre de *ac_wlan*).

UDP

A continuación, y conectado con el submódulo anterior, tenemos el encargado de modelar la capa de transporte. Esta será la capa que se conecte con el nivel de aplicación, la cual será responsable de la comunicación vehicular propiamente dicha.

Como se ha mencionado en la introducción de esta sección, este sería un módulo susceptible de ser modificado, según el interés de la simulación concreta. En el caso de utilizar el proyecto creado, la capa de transporte se implementa con el protocolo UDP.

TraCIDemo

Finalmente, el nodo debe tener una capa de aplicación que corra por encima de las anteriores. Esta es la base del interés de la simulación de las redes vehiculares, por lo que se va a explicar más en profundidad en el subapartado siguiente.

6.2.3.3 Capa de aplicación: IVC

El módulo que está más arriba en la torre de comunicación del nodo (es decir, en la capa de aplicación) será el encargado de simular el comportamiento de la comunicación intervehicular (IVC) propiamente dicha. En la siguiente Figura 43, podemos ver los componentes que implementan esta funcionalidad: módulos NED, con ficheros C++ por debajo.

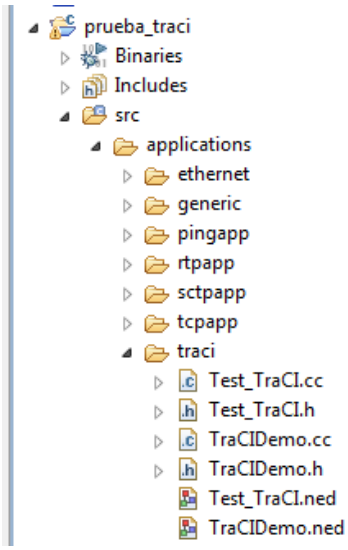


Figura 43. Ficheros de la capa de aplicación, correspondientes al IVC

TraCIDemo

En este fichero *.cc* se implementan las funciones que requiere la capa de aplicación para su actividad normal. Es decir, inicialización del módulo, envío y recepción de mensajes, configuración de la capa inferior y manejo de las actualizaciones de posición del nodo:

```
virtual void initialize( … );
void setupLowerLayer();
virtual void handleMessage( … );
virtual void handleLowerMsg( … );
virtual void sendMessage( … );
virtual void handleSelfMsg( … );
virtual void receiveChangeNotification( … );
virtual void handlePositionUpdate( … );
```

Por lo tanto, es en este componente en el que se implementa la simulación del comportamiento del IVC bajo evaluación. Es decir, el usuario que desee conocer los resultados que un determinado protocolo o sistema de comunicación intervehicular ofrecería, debe modificar este módulo en concreto.

Veamos qué funcionalidad debe implementarse en estas funciones y cuándo es llamada cada una de ellas. Todo ello, teniendo en cuenta que estas no son más que las directrices de una solución propuesta, y que el desarrollador puede crear funciones nuevas o cambiar las existentes, según su interés o su gusto.

- ***initialize***
  Debe llamarse al crear este módulo, pues su objetivo es realizar tareas de inicialización de los correspondientes a las capas inferiores, así como son la *NotificationBoard*.

- ***setupLowerLayer***
  Es la función a la que llama la anterior para definir los parámetros relativos a la capa inferior, que es la capa de transporte (puerto utilizado para la comunicación).

- ***handleMessage***
  Esta función está diseñada para el manejo de todos los mensajes que pasan por la capa de aplicación. En caso de que el mensaje sea destinado a este módulo, se llamará a la función *hadleSelfMsg*, mientras que si el destinatario es la capa inferior, se llamará a *handleLowerMsg*.

- ***handleLowerMsg***
  Esta función pasará a la capa de transporte el mensaje que se desee enviar a otro u otros nodos presentes en la red (otro u otros vehículos). Para tal fin, se debe llamar a *sendMessage*.

- ***sendMessage***
  La llamada a esta función se producirá cuando se desee enviar un mensaje a la capa inferior (desde *handleLowerMsg*, como ya se ha dicho). Aquí se construye el paquete UDP o TCP (en función del protocolo que se esté utilizando), indicando la dirección IP del destinatario. Quién es el destinatario del mensaje dependerá del funcionamiento del IVC en concreto y puede ser un parámetro que reciba esta función.

- ***handleSelfMsg***
  Se debe llamar a esta función para procesar un mensaje recibido desde otro nodo de la red. Dependiendo de las funcionalidades que se deseen implementar para la capa de aplicación en cuestión, se simulará un comportamiento determinado como reacción a la información recibida por parte de otros vehículos.

- ***reveiveChangeNotification***
  Desde esta función se manejarán los cambios producidos en la NotificationBoard (variaciones en las tablas de enrutamiento, estado de los interfaces, cambios en el canal, etc.).

- ***handlePositionUpdate***
  Cada vez que al nodo le sea comunicado un cambio de su posición (desde TraCIMobility), esta función deberá realizar las acciones que se deseen simular, por ejemplo, enviar un mensaje al resto de vehículos comunicándoselo.

Finalmente, cabe mencionar que *Test_TraCI*, que aparece en la carpeta mostrada en la Figura 43 anterior, es un programa cuya finalidad es testear los módulos de cliente TraCI de OMNeT++.

6.2.3.4 Ficheros de inicialización

Finalmente, necesitamos fijar los parámetros concretos de nuestra simulación (valor que tomarán los distintos campos en cada módulo, tiempos, etc.). Para esto, como ya se ha indicado, OMNeT++ hace uso del fichero *.ini*.

Como vemos en la Figura 44, podemos modificar este fichero desde la interfaz gráfica que nos ofrece OMNeT++:

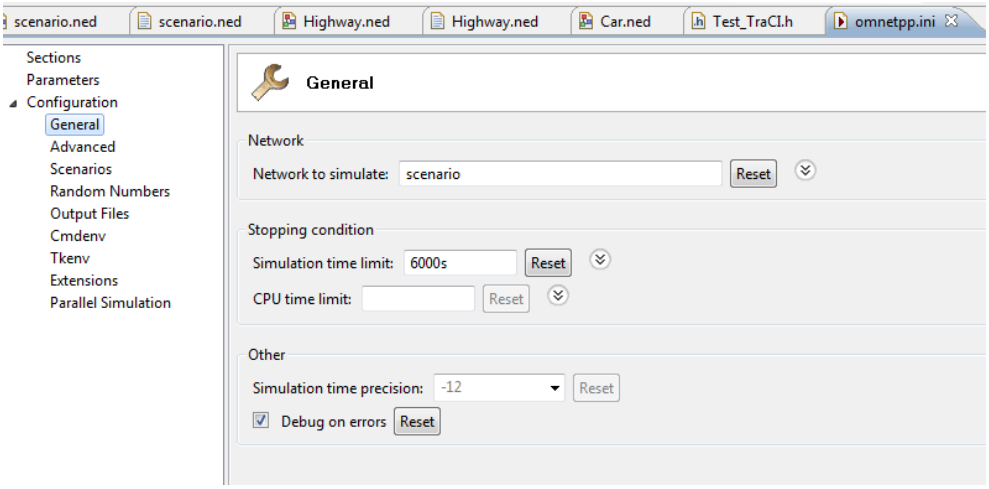


Figura 44. Modificación de parámetros del fichero de inicialización omnetpp.ini en su visualización gráfica

Asimismo, se muestra aquí el código fuente completo del fichero, que recibe el nombre de *omnetpp.ini*.

```
[General]

debug-on-errors = true

network = scenario
sim-time-limit = 6000s
print-undisposed = false
seed-0-mt = 123

cmdenv-express-mode = true
cmdenv-autoflush = true
cmdenv-status-frequency = 10000000s

tkenv-image-path = bitmaps
**.debug = true
**.coreDebug = false

*.playgroundSizeX = 10699 #[m]
*.playgroundSizeY = 7131 #[m]


# Car::TraCIMobility
*.host[*].mobility.accidentCount = 0
*.host[*].mobility.accidentStart = -1s
*.host[*].mobility.accidentDuration = -1s
*.host[*].mobility.accidentInterval = -1s


# TraCIScenarioManagerLaunchd
*.manager.updateInterval = 1s
*.manager.host = "localhost"
*.manager.port = 9999
```

```
*.manager.moduleType = "inet.examples.traci_launchd.Car"
*.manager.moduleName = "host"
*.manager.moduleDisplayString = ""
*.manager.autoShutdown = true
*.manager.margin = 25
*.manager.launchConfig = xmldoc("sumo-launchd.launch.xml")


# Network layer
**.networkLayer.ip.procDelay = 10us
**.networkLayer.arp.retryTimeout = 1s
**.networkLayer.arp.retryCount = 3
**.networkLayer.arp.cacheTimeout = 100s


# WiFi link layer
**.wlan.mgmt.frameCapacity = 10      # "maximum queue length"
**.wlan.mac.address = "auto"         # will be replaced by a generated MAC address"
**.wlan.mac.maxQueueSize = 14        # "maximum length in frames"
**.wlan.mac.bitrate = 11Mbps
**.wlan.mac.rtsThresholdBytes = 2346B   # "longer messages will be sent using RTS/CTS
**.wlan.mac.retryLimit = -1   # "maximum number of retries per message, -1: default"
**.wlan.mac.cwMinData = -1   # "contention window for normal data frames, -1: default"
**.wlan.mac.cwMinBroadcast = -1   # "contention window for bdcast msgs, -1: default"
**.wlan.radio.channelNumber = 0
**.wlan.radio.transmitterPower = 2mW
**.wlan.radio.bitrate = 11Mbps
**.wlan.radio.thermalNoise = -110dBm
**.wlan.radio.pathLossAlpha = 1.9
**.wlan.radio.snirThreshold = 3dB
**.wlan.radio.sensitivity = -85mW


# Channel Control
*.channelcontrol.carrierFrequency = 2.4GHz
*.channelcontrol.pMax = 2mW
*.channelcontrol.sat = -80dBm
*.channelcontrol.alpha = 1.9
*.channelcontrol.numChannels = 1

**.udpapp.*.vector-recording = true
**.vector-recording = true


[Config accident]

*.host[10].mobility.accidentCount = 1
*.host[10].mobility.accidentStart = 115s
*.host[10].mobility.accidentDuration = 30s
```

Como vemos, se da valor a todos los parámetros de los diferentes componentes de la simulación, anteriormente explicados. Se configuran el simulador de canal, el módulo de comunicación inalámbrica, la capa de red, así como el *TraciScenarioManager*.

A este último, entre otros parámetros, se le indica que el módulo que hará las veces de host es *Car.ned*. De igual modo, es a este componente a quien debemos mostrarle las características de la simulación de tráfico rodado, es decir, cuál será la topología y el modelo de movilidad usado (por SUMO). Para ello se utiliza el fichero *sumo-launchd.launch.xml*. La información que aporta es la que se muestra a continuación, y

será explicada en profundidad en el próximo apartado, en el que se expondrá el uso y posibilidades que nos ofrece SUMO.

```
<launch>
    <copy file="net.net.xml" />
    <copy file="routes.rou.xml" />
    <copy file="sumo.sumo.cfg" type="config" />
</launch>
```

Como comentario final, vemos que *omnetpp.ini* está estructurado en dos bloques principales: *[General]* y *[Config accident]*. Es éste último el que introduce el evento *accident* en nuestra simulación. La configuración concreta aquí mostrada, hará que el nodo 10 sufra una colisión (se detenga repentinamente) en el segundo 115 de simulación.

6.3 Redes en SUMO

6.3.1 Visión general

SUMO es en realidad un paquete de software compuesto por varios programas, los cuales son necesarios para generar la simulación de tráfico. Concretamente, cuenta con las siguientes aplicaciones [SDoc]:

- NETCONVERT:
  Es el encargado de generar las redes viarias para pasarlas al simulador.

- NETGEN:
  Generador de redes abstractas.

- JTRROUTER:
  Realiza los cálculos de las rutas utilizando porcentajes de giro en cruces.

- DFROUTER:
  Realiza los cálculos de las rutas a partir de medidas de entrada en curvas.

- DUAROUTER:
  Calcula las rutas más rápidas en la red.

- OD2TRIPS:
  Descompone matrices en rutas individualizadas para cada vehículo.

- POLYCONVERT:
  Importa puntos de interés y polígonos en diferentes formatos y los traduce a formatos que pueden ser representados por la interfaz gráfica de simulación.

- SUMO:
  Genera la simulación propiamente dicha.

- SUMO-GUI:
  Interfaz gráfica del generador de simulación.

Como se puede observar, hay tres grupos claramente diferenciados: aplicaciones orientadas al cálculo de las rutas, aplicaciones orientadas al cálculo de la red vial y finalmente, el generador de la simulación. Se puede ver esquemáticamente de forma muy clara en la siguiente Figura 45.

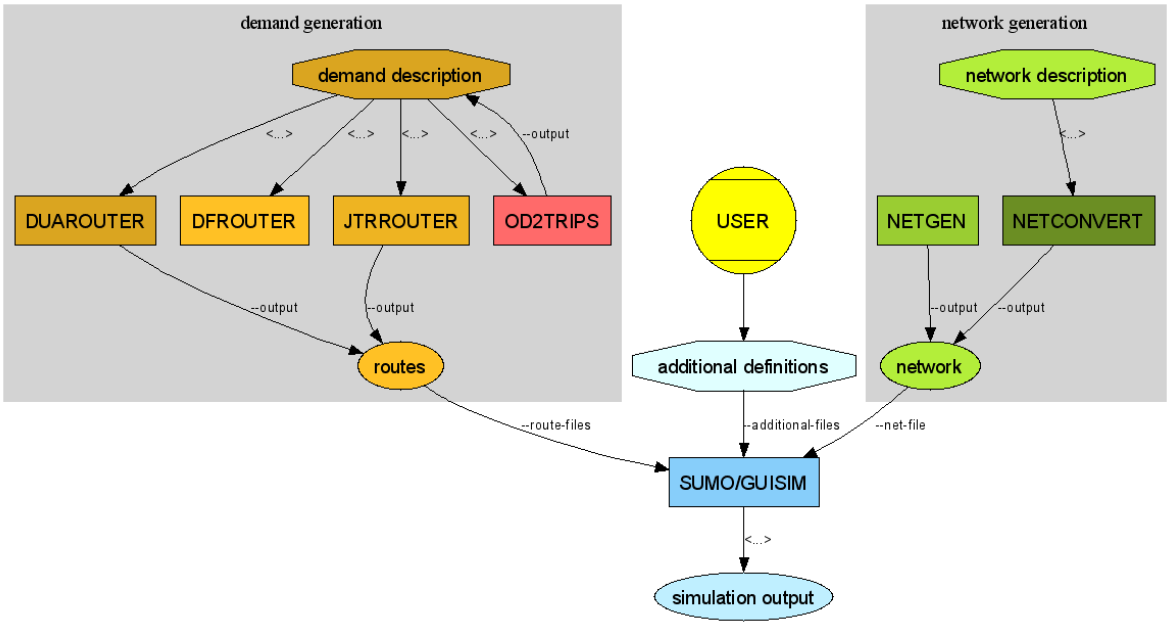


Figura 45. Estructura e interrelación de los elementos que componen una simulación SUMO

6.3.1.1 Ficheros de configuración

Puesto que la lista de opciones que ofrece SUMO es muy amplia, se decidió introducir el concepto de ficheros de configuración. Esto facilita su uso, pues no es necesario llamar todos los comandos deseados uno a uno, sino que se guarda todo el esquema en ese fichero.

Un fichero de configuración es un documento XML que tiene un elemento raíz denominado *configuration*, con los valores deseados almacenados como valores de atributo. Por ejemplo, la opción *--net-file test.net.xml* en la línea de comandos sería *<net-file value="test.net.xml"/>* en el fichero de configuración. Más adelante en este capítulo se mostrarán ejemplos que clarificarán esta funcionalidad.

Dependiendo de la aplicación (dentro del paquete SUMO) para la que vaya destinado el fichero de configuración, se estipula una nomenclatura diferente que se recomienda encarecidamente seguir:

- **.sumo.cfg*: Fichero de configuración para SUMO and GUISIM.
- **.netc.cfg*: Fichero de configuración para NETCONVERT.
- **.netg.cfg*: Fichero de configuración para NETGEN.
- **.rou.cfg*: Fichero de configuración para DUAROUTER.
- **.jtr.cfg*: Fichero de configuración para JTRROUTER
- **.df.cfg*: Fichero de configuración para DFROUTER.
- **.od2t.cfg*: Fichero de configuración para OD2TRIPS.

6.3.2 Generando la simulación

Una vez presentados los bloques y aplicaciones que van a participar en el proceso completo de la simulación, es conveniente dejar claro el orden de los pasos a seguir.

En la Figura 46 se puede ver de forma esquemática cuáles son las etapas de este proceso. Veamos, paso a paso, en qué consiste cada una de ellas en detalle.

6.3.2.1 Generación de redes viarias

Veremos a continuación cómo la herramienta NETGEN nos va a permitir generar redes viarias de tres modos: automáticamente, de forma personalizada y mediante importación.

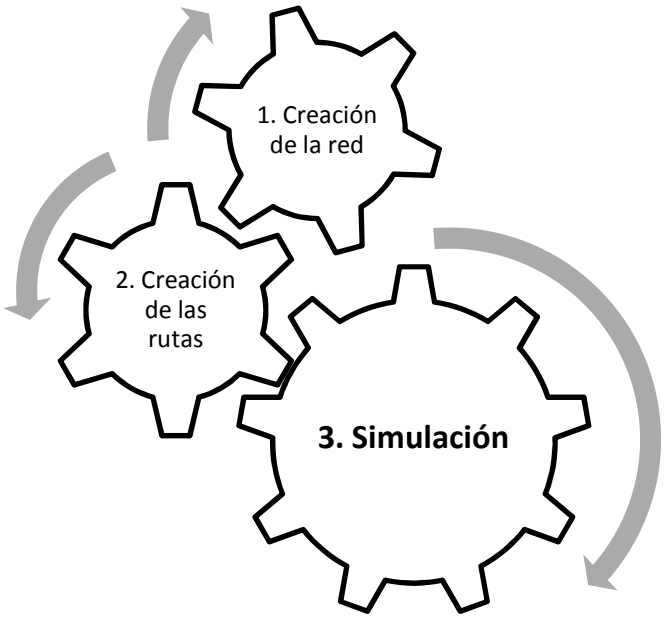


Figura 46. Creación de una simulación en SUMO

6.3.2.1.1 Generación automática de redes

El simulador va a devolver una topología de red estándar que será generada automáticamente, recibiendo, eso sí, algunos parámetros que definen a la misma por parte del usuario. Se tienen las siguientes opciones:

i. Redes de rejilla (o tipo *grid*)

Se trata de una topología con vías en sentido vertical y horizontal que se cruzan entre sí. Podremos indicar al simulador el número de cruces deseados en las coordenadas x e y, así como la distancia entre los mismos (en metros).

Las opciones correspondientes son las que se muestran a continuación.

--grid-x-number Indica el número de cruces en el eje *x*.
--grid-y-number Indica el número de cruces en el eje *y*.
--grid-number Indica el número de cruces en el eje *x* e *y* (iguales).
--grid-x-lenght Indica la distancia (metros) entre los cruces en el eje *x*.
--grid-y-lenght Indica la distancia (metros) entre los cruces en el eje *y*.
--grid-lenght Indica la distancia (metros) entre los cruces en el eje *x* e *y*.

Se dispone también de una opción para añadir vías perpendiculares al perímetro de la rejilla, de tal forma que todas las intersecciones de la red cuenten con cuatro brazos (ver Figura 52). Para ello, es necesario incluir la siguiente opción:

--attach-length

Se pude seleccionar entre dos tipos de control de tráfico en los cruces:

- *priority*: Los cruces son controlados mediante la regla de prioridad típica, es decir, al llegar a un cruce, tendrán preferencia los vehículos que aparecen por la derecha.

- *traffic_light*: El cruce se controla mediante un semáforo que irá dando paso a cada sentido de la marcha de forma intercalada.

Una vez definida la forma del *grid* y el control de las intersecciones, se deberá llamar a NETGEN para que el simulador genere la red que se le ha indicado. Para visualizar el resultado podemos cargar el archivo *.net.xml* que ha generado NETGEN (indicamos el nombre con la opción *--output-file* ó *--o*) desde la interfaz gráfica, o llamamos directamente a SUMO pasándole un archivo de configuración con extensión *.sumo.cfg* con la siguiente estructura:

```
<configuration>

        <input>
                <net-file value=mi_red.net.xml/>
        </input>
</configuration>
```

Veamos a continuación una serie de ejemplos de este tipo de redes.

1. El siguiente comando dará como resultado una red automática cuadrada de rejilla con un total de 4 cruces en cada uno de los ejes, así como una distancia de 150 metros de separación entre los mismos:

   *netgen --grid-net --grid-number=4 --grid-length=150 *
   --output-file=red_rejilla_1.net.xml

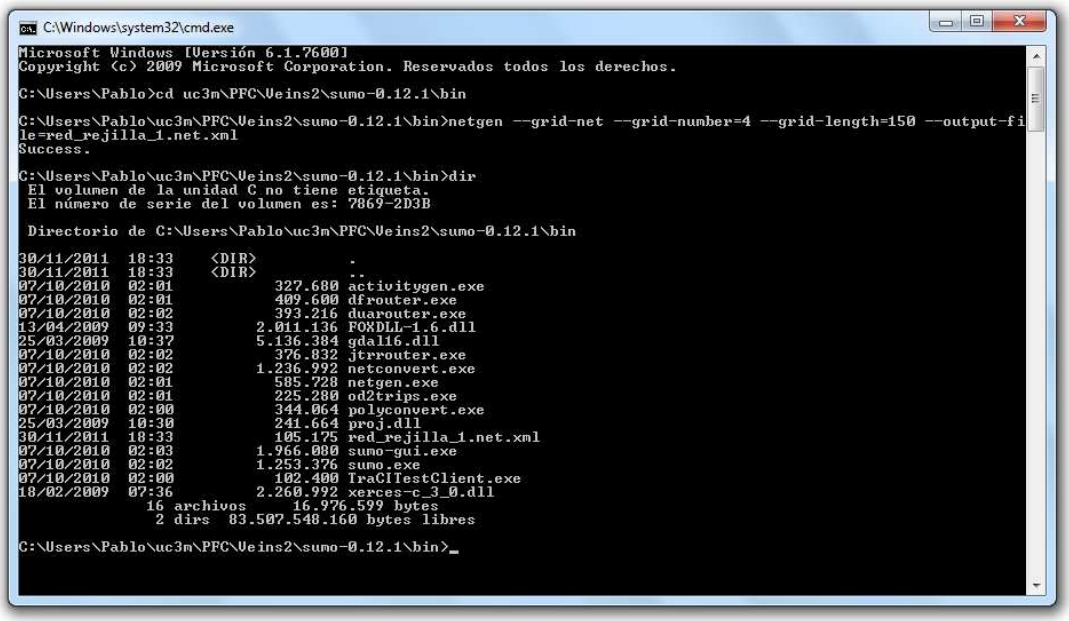


Figura 47. Creación de una red tipo grid mediante NETGEN

El resultado mostrado por la interfaz gráfica de SUMO puede observarse en la siguiente Figura 48.

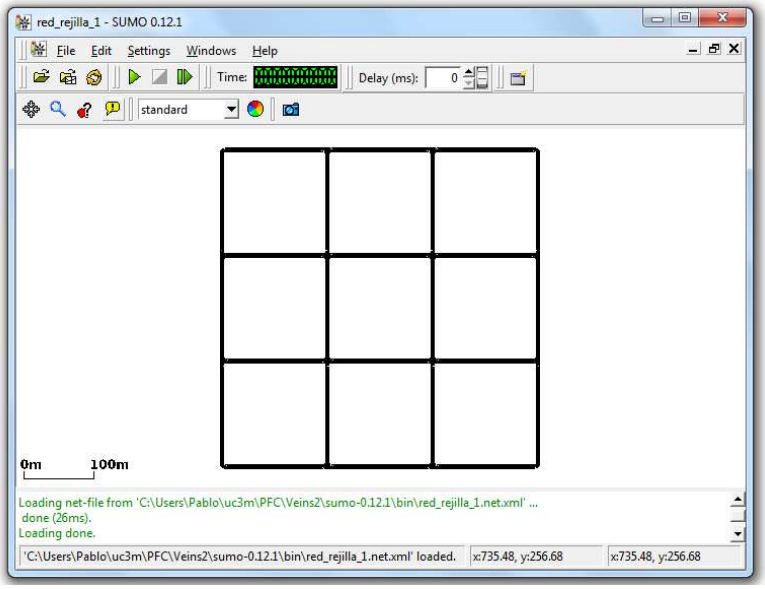


Figura 48. Red tipo rejilla (ejemplo 1)

Haciendo zoom en la red, vemos cómo son representadas las intersecciones (Figura 49).

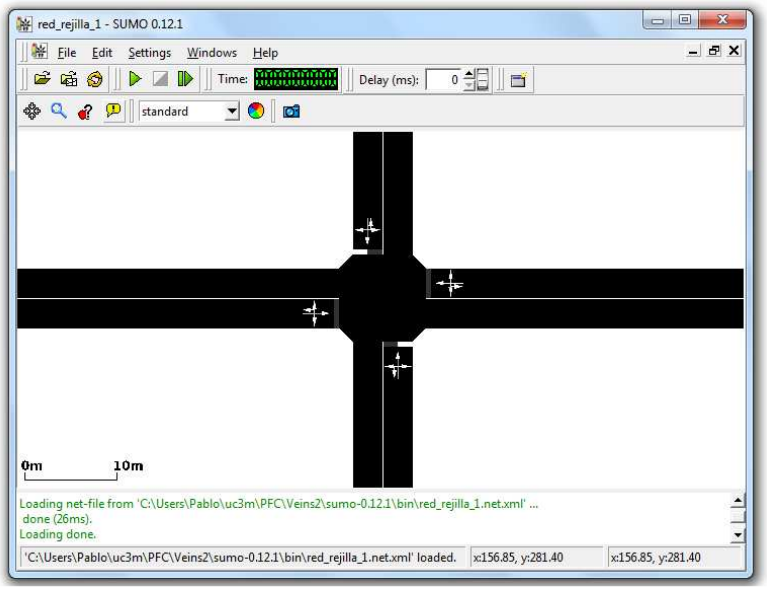


Figura 49. Red tipo rejilla (ejemplo 1). Ampliación de una intersección

2. A continuación veamos cómo se generaría una red similar, pero en este caso, con las intersecciones controladas por semáforos. Véase que para ello se debe añadir el tipo de control de cruces precedido por *-junction* (o también *-j*):

*netgen --grid-net --grid-number=8 --grid-length=100 –j traffic_light *
--output-file=red_rejilla_2.net.xml

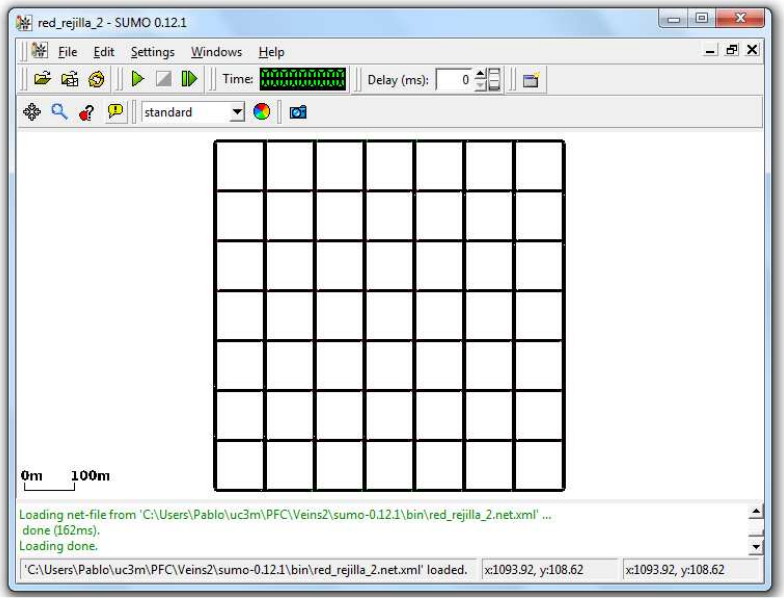


Figura 50. Red tipo rejilla con intersecciones controladas por semáforos (ejemplo 2)

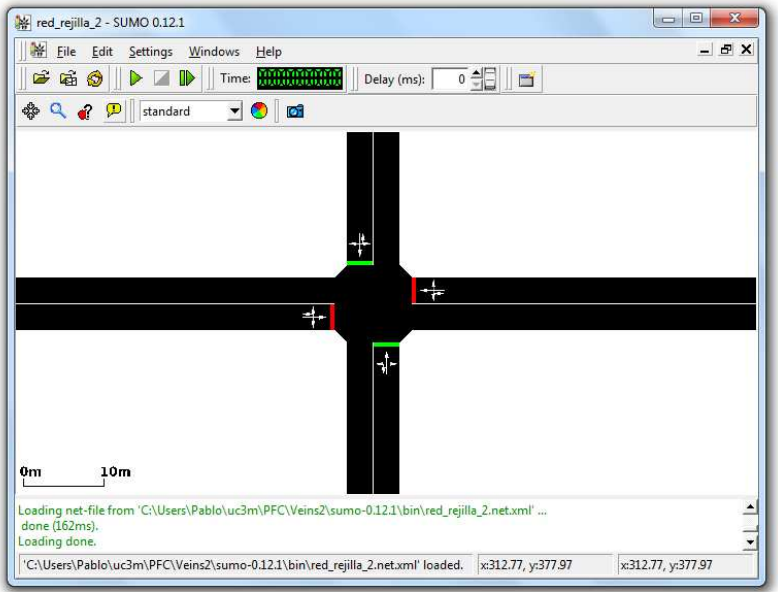


Figura 51. Red tipo rejilla con intersecciones controladas por semáforos. Detalle de un cruce (ejemplo 2)

3. Para finalizar, veamos cómo aplicar el resto de opciones que se han presentado. Concretamente, el siguiente ejemplo generará una red con 5 cruces en el eje x y 7 en él y, separados 50 y 70 metros en cada eje respectivamente. Los cruces serán controlados mediante semáforos y haremos que haya cuatro brazos de tráfico por cada uno:

*netgen --grid-net --grid-x-number=5 --grid-y-number=7 --grid-x-length=50 *
*--grid-y-length=70 –j traffic_light --attach-length=60 *
--output-file=red_rejilla_3.net.xml

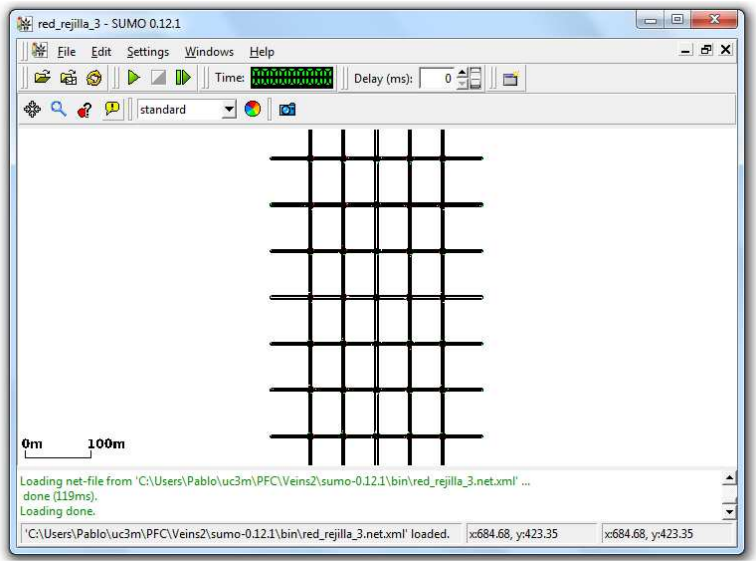


Figura 52. Red tipo rejilla con múltiples modificaciones (ejemplo 3)

ii. Redes de araña (o tipo *spider*)

Este segundo tipo de redes automáticas debe su nombre al hecho de tener la forma de una tela de araña. Una red de este tipo se define por el número de ejes por los que está formada, el número de círculos concéntricos que se superponen a dichos ejes y la distancia entre los mismos (por defecto, serán 13, 20 y 100 metros, respectivamente). Las opciones para generarla son las siguientes:

--spider-arm-number Indica el número de ejes.
--spider-circlenumber (ó *--circles*) Indica el número de círculos concéntricos.
--spider-space-rad (ó *--radius*) Indica la distancia (m) de separación entre los círculos.

Cabe destacar que el centro de la red (lugar del que parten los ejes o brazos) supone un punto de conflicto, pues se trata de una intersección entre un gran número de vías. Esta unión es tratada por el simulador como una intersección controlada por el paradigma típico de preferencia de paso.

Tal y como se hizo para las redes tipo *grid*, se muestran ahora unos ejemplos prácticos de generación de redes de araña.

1. Veamos el resultado de generar una red tipo *spider* con 8 ejes y 5 círculos. Se fija la distancia de separación entre los círculos en 75 metros. El código, por tanto, será el siguiente:

   *netgen --spider-net --spider-arm-number=8 --spider-circle-number=5 *
   --spider-space-rad=75 --output-file=red_spider_1.net.xml

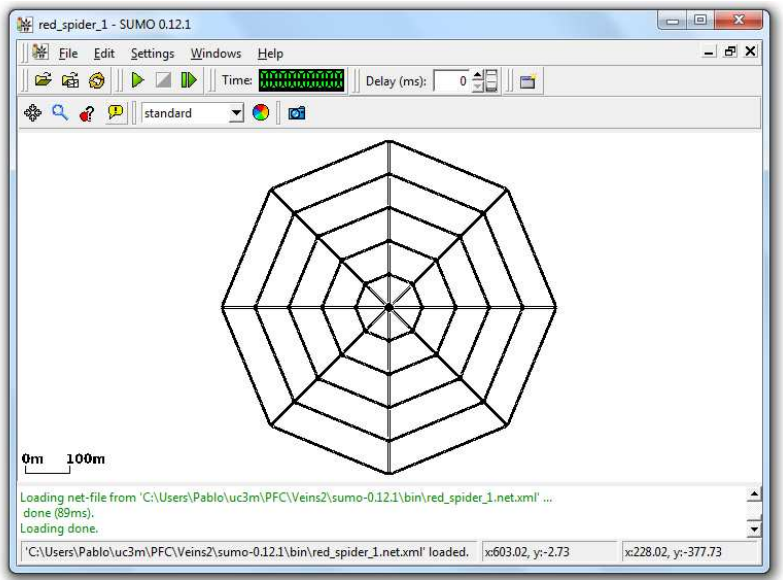


Figura 53. Red tipo araña (ejemplo 1)

2. Otro ejemplo que puede ser de gran utilidad es el correspondiente a la red del ejemplo anterior, pero en este caso, sin punto central de confluencia de las ramas o ejes. El código correspondiente es el que se muestra a continuación:

*netgen --spider-net –spider-arm-number=7 --spider-circle-number=5 *
--spider-space-rad=75 --spider-omit-center --output-file=red_spider_2.net.xml

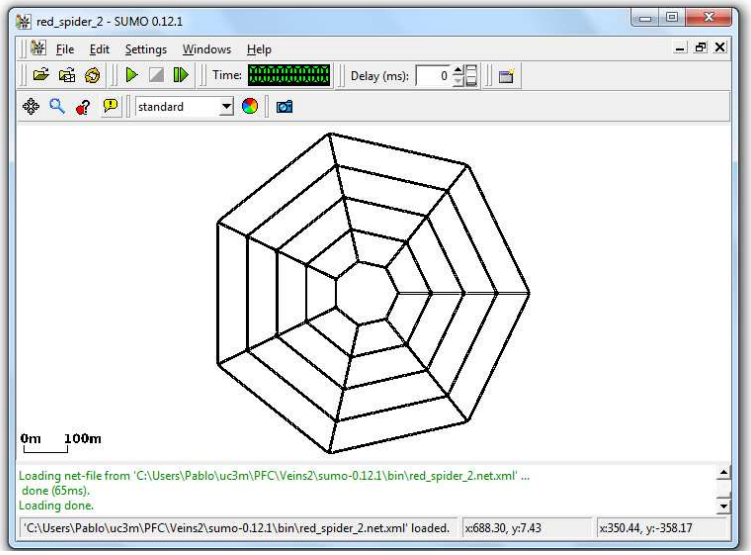


Figura 54. Red tipo araña sin centro (ejemplo 2)

iii. Redes aleatorias

Existe es una muy buena opción para generar redes de carácter aleatorio. Para esta modalidad es necesario introducir una serie de valores para los siguientes parámetros, lo cual dará lugar a una red u otra:

--rand-max-distance
--rand-min-distance
--rand-min-angle
--rand-num-tries
--rand-connectivity
--rand-neighbor-dist

A continuación se muestra un ejemplo práctico de este tipo de redes. Concretamente, el siguiente comando generará una red aleatoria con una separación máxima entre segmentos de 200 metros. Además, indicaremos al simulador que deseamos que realice un total de 500 iteraciones:

*netgen --random-net --output-file=red_aleatoria.net.xml --rand-iterations=500 *
--rand-max-distance=200 --rand-min-angle=1

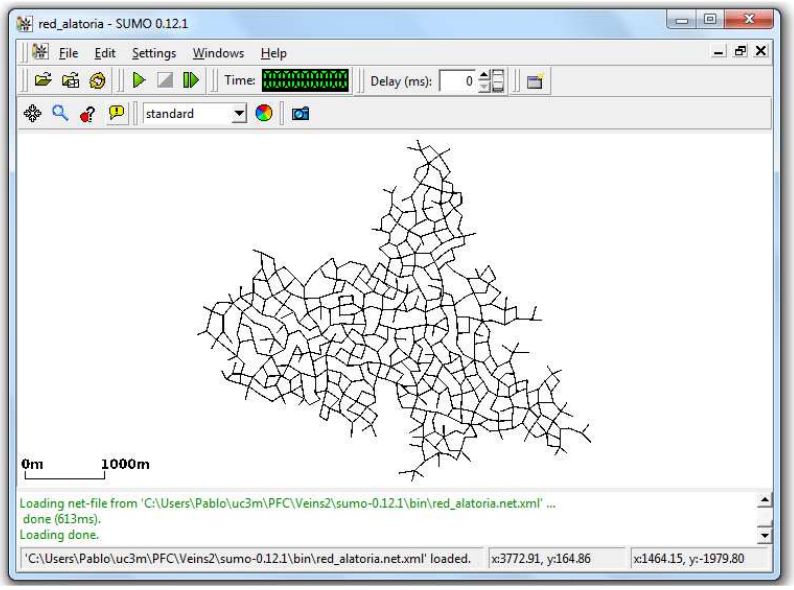


Figura 55. Red aleatoria

6.3.2.1.2 Generación de redes personalizadas

Éste es el segundo modo en el que el simulador nos permite generar las redes. Permite al usuario realizar un diseño personal de la topología, según sus necesidades.

Como mínimo, es necesario que se creen dos ficheros: uno que defina los nodos de la red y otro los enlaces que los unen (en la terminología en inglés que usa el simulador, serían *nodes* y *edges*, respectivamente). No obstante, existe la posibilidad de incluir dos ficheros más, uno de ellos determina los atributos de los enlaces, es decir, de los carriles de las carreteras; y otro, su tipo. La nomenclatura que determina el simulador es *connections* para el primero y *types* para el último.

Veamos cómo es necesario que se creen estos cuatro ficheros (o al menos, los dos principales) con la descripción de la red, para posteriormente pasárselos a la herramienta NETCONVERT. Para hacer esta llamada a NETCONVERT, los comandos a ejecutar serán los siguientes.

- Versión básica:

*netconvert --xml-node-files=MisNodos.nod.xml --xml-edge-files=MisEnlaces.edg.xml *

--output-file=MiRed.net.xml

- Versión completa indicando *types* y *connections*:

*netconvert --xml-node-files=Mis_Nodos.nod.xml --xml-edge-files=Mis_Enlaces.edg.xml *
*--xml-connection-files=Mis_Conexiones.con.xml --xml-type-files=Mis_Tipos.typ.xml *
--output-file=Mi_Red.net.xml

De forma esquemática:

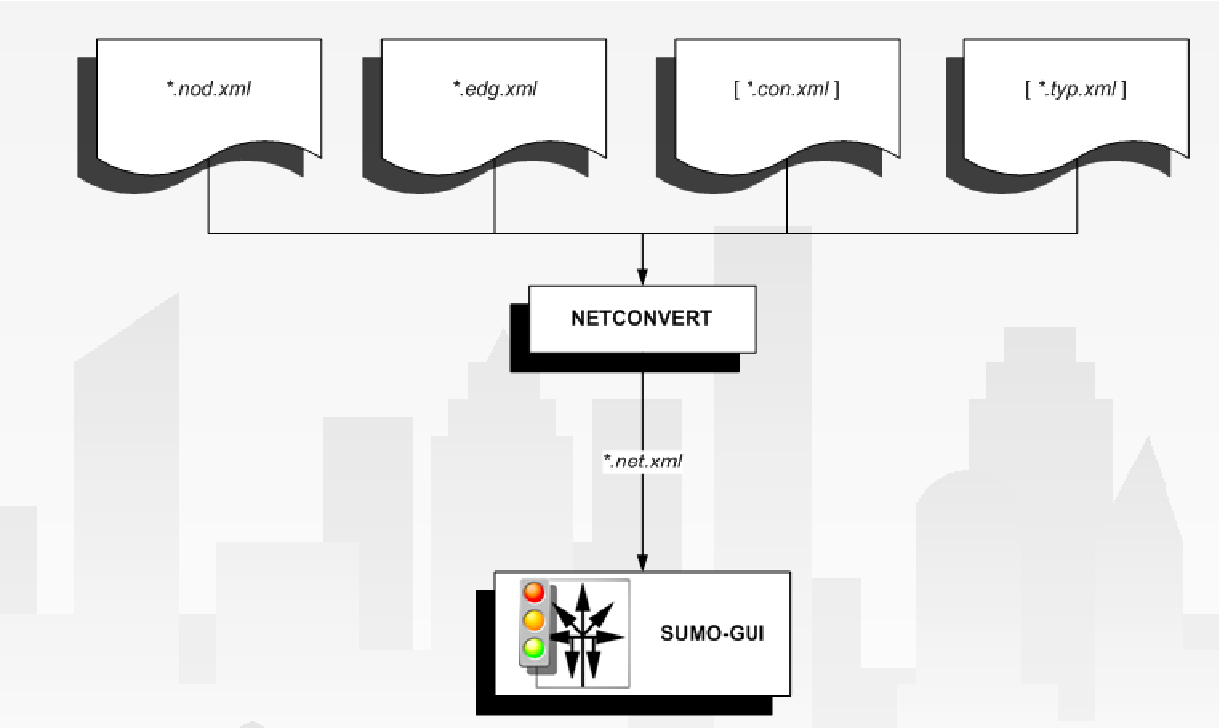


Figura 56. Esquema para la generación de topologías de red viaria personalizadas y su posterior representación en SUMO

A continuación se explica qué información contiene cada uno de los ficheros ya mencionados.

i. Nodos

En el esquema de la red, un nodo representa la unión de varias vías, es decir, una intersección.

Como ya se ha dicho, el fichero que incluirá la definición de los nodos debe tener extensión *.nod.xml*. La sentencia para cada nodo debe ser la siguiente:

<node id="<STRING>" x="<FLOAT>" y="<FLOAT>" [type="<TYPE>"]/>

Donde:

- *node id*: identificador del nodo.
- *x*: posición del nodo en el eje *X* (en metros).
- *y*: posición del nodo en el eje *Y* (en metros).
- *type*: indica el tipo de nodo y es opcional (puede ser *"priority", "traffic_light"* o *"right_before_left"*)

Las coordenadas deben indicarse en metros y con referencia al centro de la red. Es decir, como se muestra en la siguiente figura.

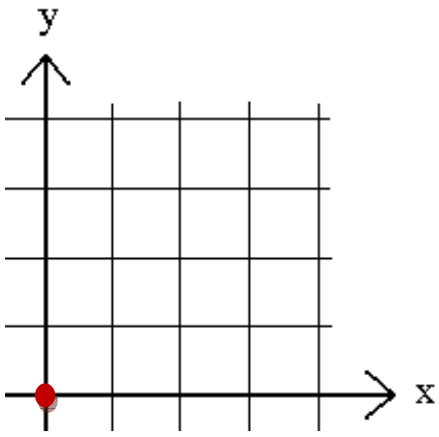


Figura 57. Sistema de coordenadas para la ubicación de un nodo, donde el punto rojo es el origen/centro de la red

Por otra parte, el tipo asociado al nodo (parámetro *type*) permite definir cómo será el funcionamiento de la intersección que representa. Se ofrecen tres opciones, aunque en caso de dejarlo en blanco, NETCONVERT elegirá una de ellas aleatoriamente:

- *priority*: los vehículos esperarán a que hayan cruzado la intersección los que vienen por la derecha.
- *traffic_light*: la intersección es controlada mediante un semáforo.
- *right_before_left*: los vehículos dejarán pasar a los que vienen por su derecha.

Se muestra a continuación un ejemplo práctico [SUMO], denominado *Mi_Red*, de representación de una topología de red concreta. Veamos cómo plasmarla en ficheros que NETCONVERT entienda.

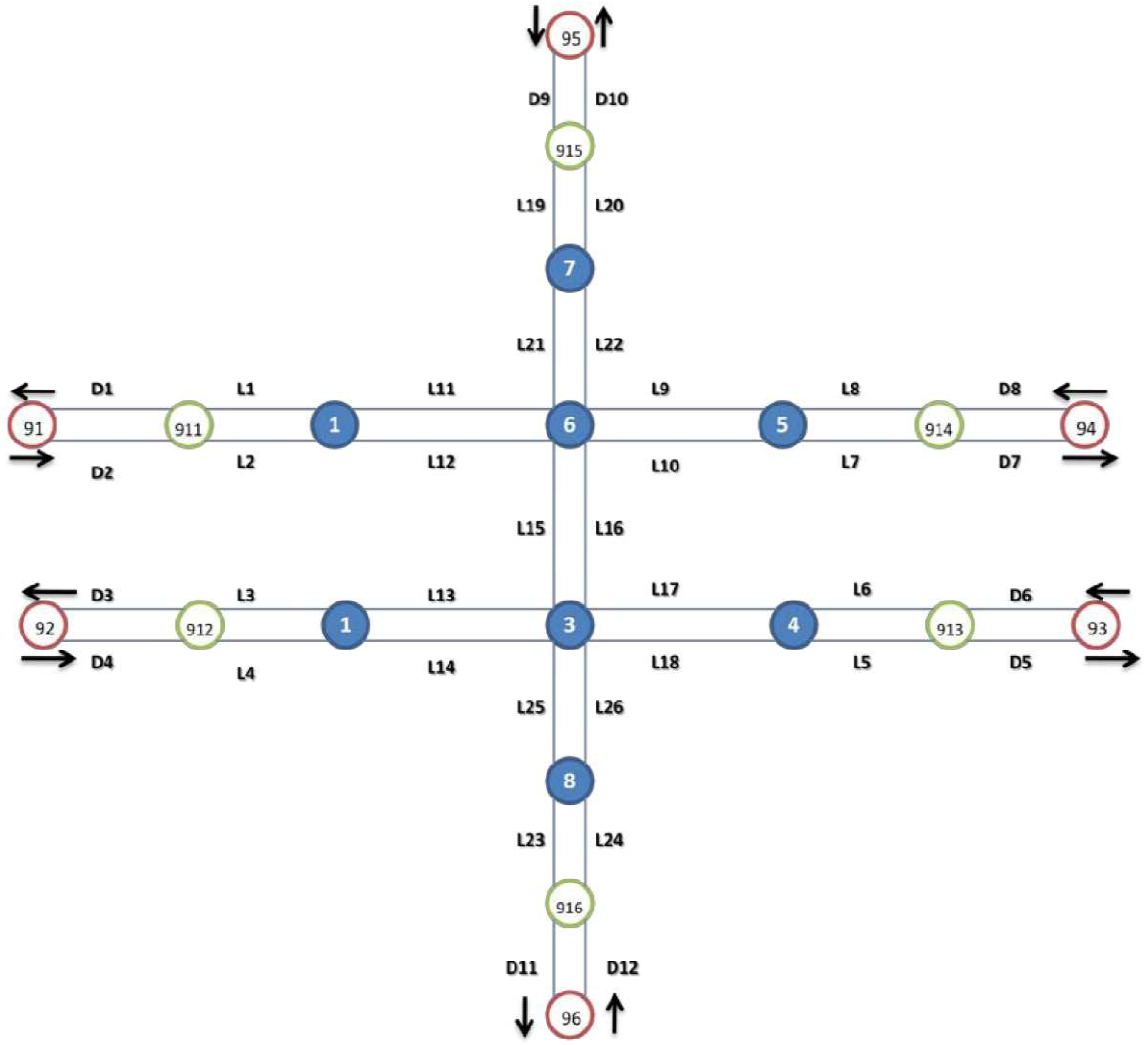


Figura 58. Esquema de la red Mi_Red, que se usará como caso práctico de estudio

En primer lugar, necesitamos definir la topología de los nodos mediante un fichero *Mis_Nodos.nod.xml*. Será de la siguiente forma:

```
<nodes>
<node id="91" x="-1000.0" y="1000.0" />
<node id="92" x="-1000.0" y="0.0" />
<node id="93" x="3000.0" y="0.0" />
<node id="94" x="+3000.0" y="+1000.0" />
<node id="95" x="+1000.0" y="+3000.0" />
<node id="96" x="+1000.0" y="-3000.0" />
<node id="911" x="-500.0" y="+1000.0" />
<node id="912" x="-500.0" y="0.0" />
```

```
<node id="913" x="+2500.0" y="0.0" />
<node id="914" x="2500.0" y="+1000.0" />
<node id="915" x="+1000.0" y="+2500.0" />
<node id="916" x="+1000.0" y="-2500.0" />
<node id="1" x="0.0" y="+1000.0" />
<node id="2" x="0.0" y="0.0" />
<node id="3" x="+1000.0" y="0.0" type="traffic_light" />
<node id="4" x="+2000.0" y="0.0" />
<node id="5" x="+2000.0" y="+1000.0" />
<node id="6" x="+1000.0" y="+1000.0" type="traffic_light" />
<node id="7" x="+1000.0" y="+2000.0" />
<node id="8" x="+1000.0" y="-1000.0" />
</nodes>
```

ii. Enlaces

La descripción de los enlaces puede realizarse de dos formas distintas. La primera opción se basa en los nodos previamente definidos y simplemente indica desde cuál parte el enlace (*fromnode*) y en cuál finaliza (*tonode*). La sentencia es como sigue:

<edge id="<STRING>"fromnode="<NODE_ID>" tonode="<NODE_ID>"/>

La segunda opción consiste en indicar la pareja de coordenadas de origen del nodo (*xfrom, yfrom*) y la de destino (*xto, yto*). En este caso, de no existir ningún nodo en una de las posiciones indicadas, se creará automáticamente.

<edge id="<STRING>" xfrom="<FLOAT>" yfrom="<FLOAT>" xto="<FLOAT>" \ yto="<FLOAT>") />

Cabe notar que los enlaces tienen siempre carácter unidireccional, como se desprende de su definición (pues se indica un origen y un destino). No obstante, podemos modificar algunas de sus propiedades. Esto puede hacerse mediante la definición de un *tipo* o indicándolo en el propio archivo de definición de enlace. En el caso de optar por la segunda opción, se deben incluir los siguientes atributos en la sentencia de definición del enlace:

nolanes="<INT>" speed="<FLOAT>" priority="<UINT>" length="<FLOAT>" \ shape="<2D_POINT> [<2D_POINT>] spread_type="center"*

Donde:

- *nolanes*: número entero indicador del número de carriles.

- *speed*: velocidad máxima permitida en el enlace en *m/s*.
- *priority*: entero indicador de la prioridad del enlace.
- *length*: número decimal que indica la longitud en metros del enlace.
- *shape*: indica la forma del enlace mediante la definición de uno o más puntos por los que debe pasar el mismo.
- *spread_type*: indicador de cuál será la distribución de los carriles dentro del enlace. Puede ser "*center*" (para una distribución uniforme) o "*right*" (para alinearlos a la derecha).

Por otra parte, si se decide definir un *tipo*, no hay más que crear un fichero *.typ.xml* en el cual se detalle el tipo de vía de la simulación con la estructura indicada anteriormente. Es decir:

*<type id="<STRING>" nolanes="<INTEGER>" speed="<FLOAT> *
"priority="<UINT>"/>

Cabe mencionar que aunque se cree un fichero *type*, siempre existe la posibilidad de sobrescribir cualquiera de los parámetros desde la línea de comandos.

Como continuación de la generación de la red de ejemplo, se define el siguiente archivo *type Mis_Tipos.typ.xml*:

```
<types>
<type id="a" priority="3" nolanes="3" speed="15.0" />
<type id="b" priority="3" nolanes="2" speed="15.0" />
<type id="c" priority="2" nolanes="3" speed="15.0" />
<type id="d" priority="1" nolanes="2" speed="20.0" />
</types>
```

Y el consiguiente fichero de enlaces *Mis_Enlaces.edg.xml*:

```
<edges>
<edge id="D1" fromnode="911" tonode="91" type="a" />
<edge id="D2" fromnode="91" tonode="911" type="b" />
<edge id="D3" fromnode="912" tonode="92" type="a" />
<edge id="D4" fromnode="92" tonode="912" type="b" />
<edge id="D5" fromnode="913" tonode="93" type="a" />
<edge id="D6" fromnode="93" tonode="913" type="b" />
<edge id="D7" fromnode="914" tonode="94" type="a" />
<edge id="D8" fromnode="94" tonode="914" type="b" />
<edge id="D9" fromnode="95" tonode="915" type="b" />
<edge id="D10" fromnode="915" tonode="95" type="a" />
```

```
<edge id="D12" fromnode="96" tonode="916" type="d" />
<edge id="D11" fromnode="916" tonode="96" type="a" />
<edge id="L1" fromnode="1" tonode="911" type="a" />
<edge id="L2" fromnode="911" tonode="1" type="b" />
<edge id="L3" fromnode="2" tonode="912" type="a" />
<edge id="L4" fromnode="912" tonode="2" type="b" />
<edge id="L5" fromnode="4" tonode="913" type="a" />
<edge id="L6" fromnode="913" tonode="4" type="b" />
<edge id="L7" fromnode="5" tonode="914" type="a" />
<edge id="L8" fromnode="914" tonode="5" type="b" />
<edge id="L9" fromnode="5" tonode="6" type="a" />
<edge id="L10" fromnode="6" tonode="5" type="a" />
<edge id="L11" fromnode="6" tonode="1" type="a" />
<edge id="L12" fromnode="1" tonode="6" type="a" />
<edge id="L13" fromnode="3" tonode="2" type="a" />
<edge id="L14" fromnode="2" tonode="3" type="a" />
<edge id="L15" fromnode="6" tonode="3" type="c" />
<edge id="L16" fromnode="3" tonode="6" type="c" />
<edge id="L17" fromnode="4" tonode="3" type="a" />
<edge id="L18" fromnode="3" tonode="4" type="a" />
<edge id="L19" fromnode="915" tonode="7" type="b" />
<edge id="L20" fromnode="7" tonode="915" type="a" />
<edge id="L21" fromnode="7" tonode="6" type="a" />
<edge id="L22" fromnode="6" tonode="7" type="a" />
<edge id="L23" fromnode="8" tonode="916" type="a" />
<edge id="L24" fromnode="916" tonode="8" type="b" />
<edge id="L25" fromnode="3" tonode="8" type="a" />
<edge id="L26" fromnode="8" tonode="3" type="a" />
</edges>
```

iii. Conexiones

Para finalizar, únicamente resta definir cómo se desea que sean las conexiones entre los distintos enlaces. En otras palabras, la forma en que se conectan los distintos carriles entre sí en los puntos de unión de los enlaces, es decir, en los nodos.

No obstante, en caso de dejar alguna conexión sin definir, NETCONVERT automáticamente calcula la información que falta basándose en métodos heurísticos.

La sintaxis es de la forma que sigue:

<connection from="Enlace_A" to="Enlace_B" fromLane="X" toLane="Y" />

Donde *X* es el carril del enlace *Enlace_A* que se desea conectar con el carril *Y* del enlace *Enlace_B*.

Para nuestro caso definiremos el siguiente fichero *Mis_Conexiones.con.xml*.

```
<connections>
<connection from="L2" to="L12" fromLane="0" toLane="0" />
<connection from="L2" to="L12" fromLane="0" toLane="1" />
<connection from="L2" to="L12" fromLane="1" toLane="2" />
<connection from="L4" to="L14" fromLane="0" toLane="0" />
<connection from="L4" to="L14" fromLane="1" toLane="1" />
<connection from="L4" to="L14" fromLane="1" toLane="2" />
<connection from="L19" to="L21" fromLane="0" toLane="0" />
<connection from="L19" to="L21" fromLane="0" toLane="1" />
<connection from="L19" to="L21" fromLane="1" toLane="2" />
<connection from="L24" to="L26" fromLane="0" toLane="0" />
<connection from="L24" to="L26" fromLane="0" toLane="1" />
<connection from="L24" to="L26" fromLane="1" toLane="2" />
<connection from="L9" to="L11" fromLane="0" toLane="0" />
<connection from="L9" to="L11" fromLane="1" toLane="1" />
<connection from="L9" to="L11" fromLane="1" toLane="2" />
<connection from="L9" to="L15" fromLane="1" toLane="2" />
<connection from="L9" to="L15" fromLane="2" toLane="2" />
<connection from="L9" to="L22" fromLane="0" toLane="0" />
<connection from="L9" to="L22" fromLane="1" toLane="2" />
<connection from="L16" to="L10" fromLane="0" toLane="0" />
<connection from="L16" to="L10" fromLane="1" toLane="1" />
<connection from="L16" to="L10" fromLane="1" toLane="2" />
<connection from="L16" to="L11" fromLane="2" toLane="2" />
<connection from="L12" to="L15" fromLane="0" toLane="0" />
<connection from="L12" to="L15" fromLane="1" toLane="1" />
<connection from="L12" to="L10" fromLane="1" toLane="0" />
<connection from="L12" to="L10" fromLane="1" toLane="1" />
<connection from="L12" to="L10" fromLane="2" toLane="2" />
<connection from="L14" to="L16" fromLane="1" toLane="1" />
<connection from="L14" to="L16" fromLane="1" toLane="0" />
<connection from="L14" to="L16" fromLane="2" toLane="2" />
<connection from="L14" to="L18" fromLane="0" toLane="0" />
<connection from="L14" to="L18" fromLane="1" toLane="1" />
<connection from="L14" to="L18" fromLane="1" toLane="2" />
<connection from="L17" to="L16" fromLane="0" toLane="0" />
<connection from="L17" to="L16" fromLane="1" toLane="1" />
<connection from="L17" to="L16" fromLane="1" toLane="2" />
<connection from="L17" to="L13" fromLane="1" toLane="0" />
<connection from="L17" to="L13" fromLane="1" toLane="1" />
<connection from="L17" to="L13" fromLane="2" toLane="2" />
<connection from="L17" to="L25" fromLane="1" toLane="2" />
<connection from="L17" to="L25" fromLane="2" toLane="2" />
</connections>
```

Veamos el resultado. En primer lugar debemos pasar a *NETCONVERT* todos los ficheros generados de la forma ya indicada, como se hace en la Figura 59.

Figura 59. Llamada a NETCONVERT

Una vez comprobado que la generación ha sido satisfactoria (el resultado es *Success*), abrimos la red generada desde la interfaz gráfica de SUMO, obteniéndose el resultado que se representa en la siguiente Figura 60.

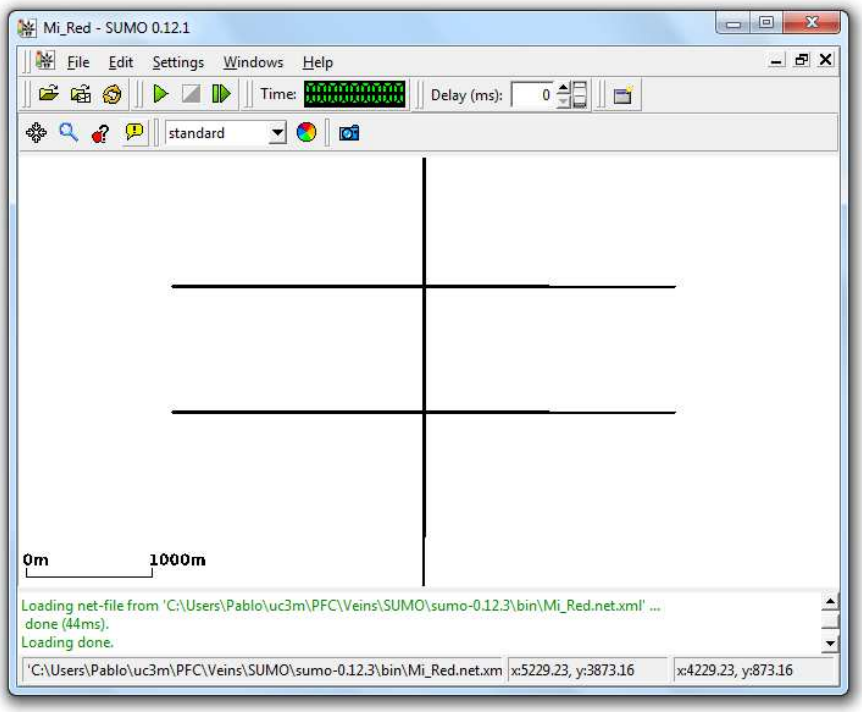


Figura 60. Representación de la red generada Mi_Red en la interfaz gráfica

Se comprueba a continuación que el resultado obtenido efectivamente se corresponde al diseño indicado en los ficheros anteriormente desarrollados. Veamos ampliados varios puntos importantes de nuestra red. En primer lugar se muestra el nodo 5 (Figura 61), en el que se puede ver la unión de los cuatro enlaces que en él se cruzan, así como las conexiones entre los diferentes carriles (obsérvense las direcciones que indican las flechas en el cruce).

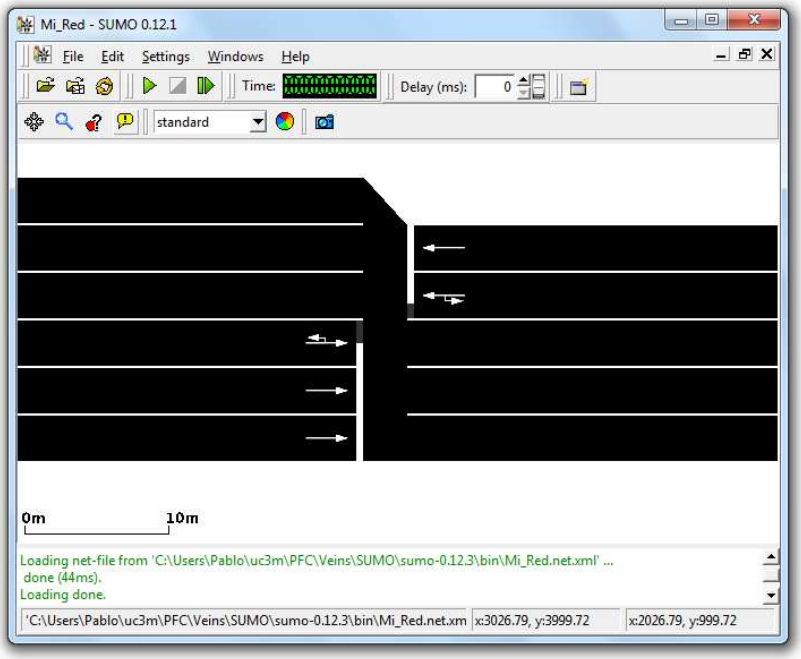


Figura 61. Zoom de la parte de red correspondiente al nodo 5

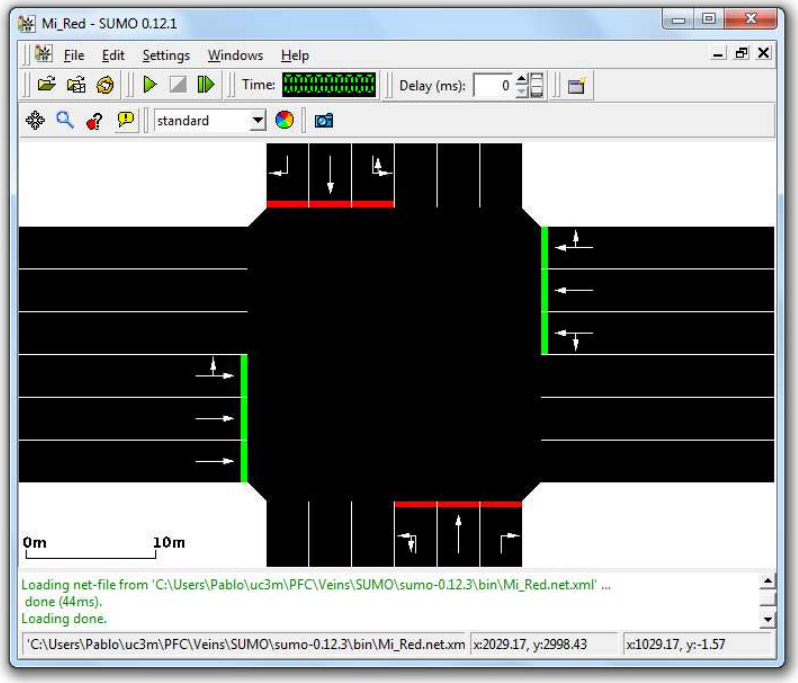


Figura 62. Zoom de la parte de la red correspondiente al nodo 6

También se puede observar en la segunda figura de esta página (Figura 62) el resultado correspondiente al nodo 6. Se trata de un cruce de 8 enlaces, con 24 carriles en total, en el que las flechas nos indican cómo son las conexiones entre los mismos.

Para finalizar se muestra la unión de varios enlaces en los que no existe una transición en el número de carriles. Como se ve en la Figura 63 a continuación, la

definición de un nodo (en este caso el nodo *916*) puede servirnos únicamente para permitir la existencia de unas determinadas conexiones entre carriles.

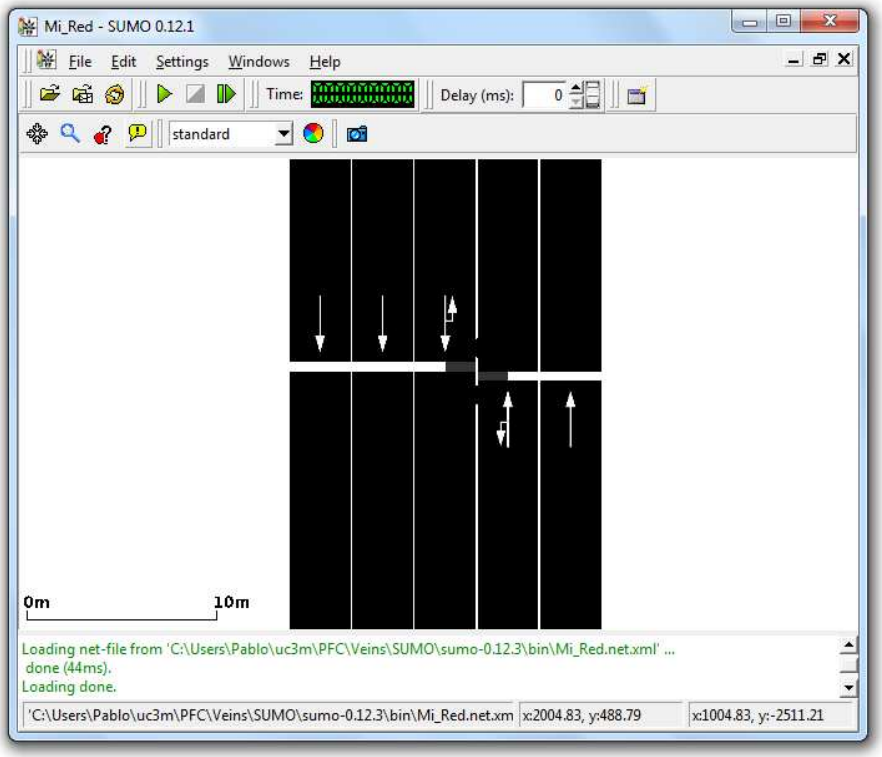


Figura 63. Zoom de la parte de red correspondiente al nodo 916

6.3.2.1.3 Generación de redes por importación

SUMO nos ofrece la posibilidad de importar topologías reales de red para nuestra simulación, lo cual supone una importante ventaja, pues se trata de una posibilidad muy interesante. El usuario podrá usar escenarios reales y concretos (de interés para su aplicación) de cara a estudiar el comportamiento de un determinado IVC.

El simulador es capaz de importar redes desde varias fuentes, no obstante, nos centraremos aquí en una de las principales, por tratarse de, quizás, la más completa: OpenStreetMap.

- **OpenStreetmap**

Se trata de un proyecto colaborativo cuyo fin es crear mapas libres y editables y ponerlos a disposición de todos los usuarios. El usuario puede acceder a él en [OSMap]. Nos permite obtener mapas de entre una base de datos realmente extensa, obteniendo un

nivel de detalle muy interesante, como el que se observa en el ejemplo de la Figura 64 mostrada a continuación.

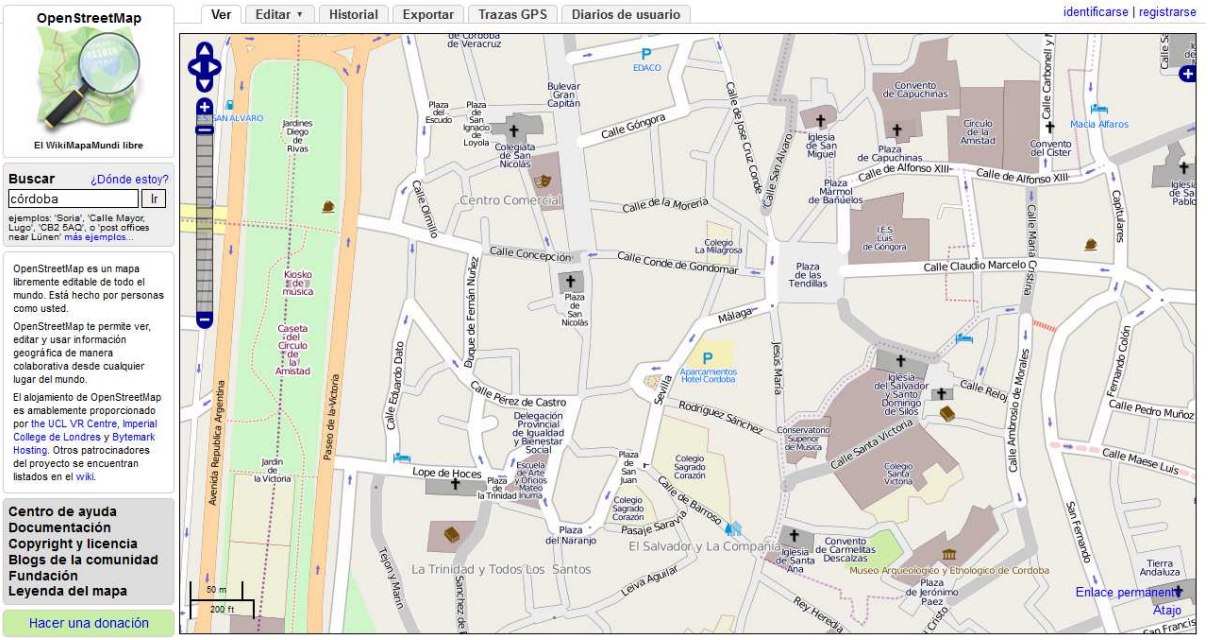


Figura 64. Captura de OpenStreetMap correspondiente a una parte del centro urbano de la ciudad de Córdoba

Desde esta misma herramienta web se nos ofrece la posibilidad de realizar la exportación del mapa. En nuestro caso, nos interesará el formato XML, pues es este tipo de ficheros el que SUMO es capaz de procesar.

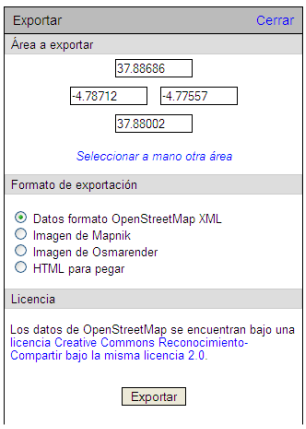


Figura 65. Proceso de exportación con formato xml

Como utilidad complementaria para OpenStreetMap, cabe mencionar la aplicación JOSM (véase [JOSM]). Se trata de una extensión desarrollada en Java, que nos permite

realizar modificaciones sobre el mapa importado desde OpenStreetMap, tales como direcciones permitidas en las vías, nuevas calles, etc.

Podemos ver una captura de nuestro mapa de ejemplo, visto en JOSM, en la siguiente Figura 66.

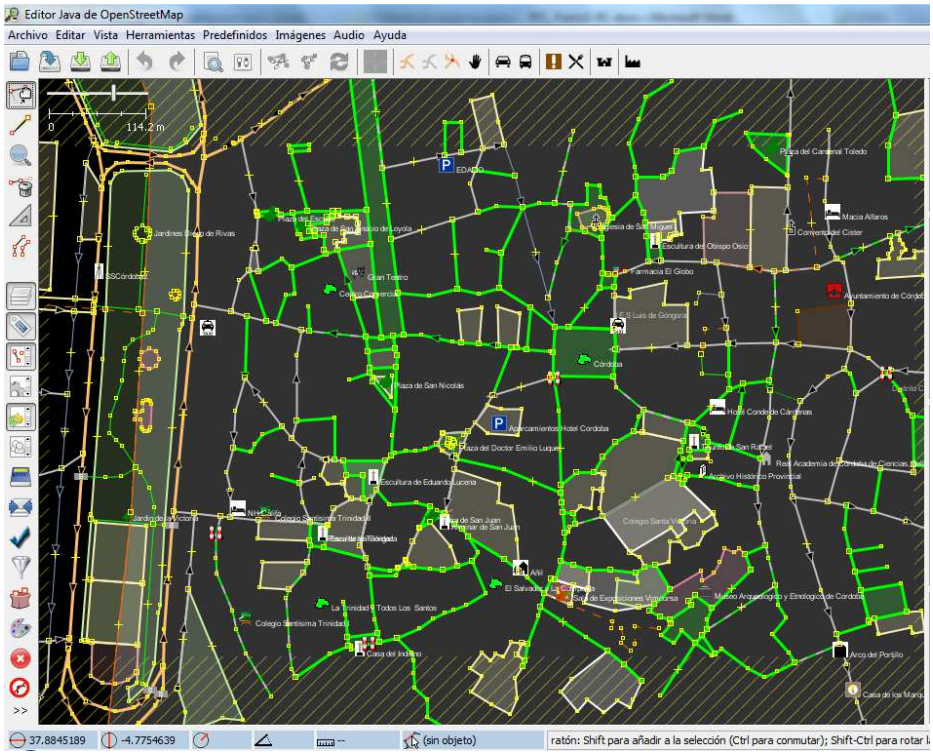


Figura 66. Mapa exportado desde OpenStreetMap, editado en JOSM

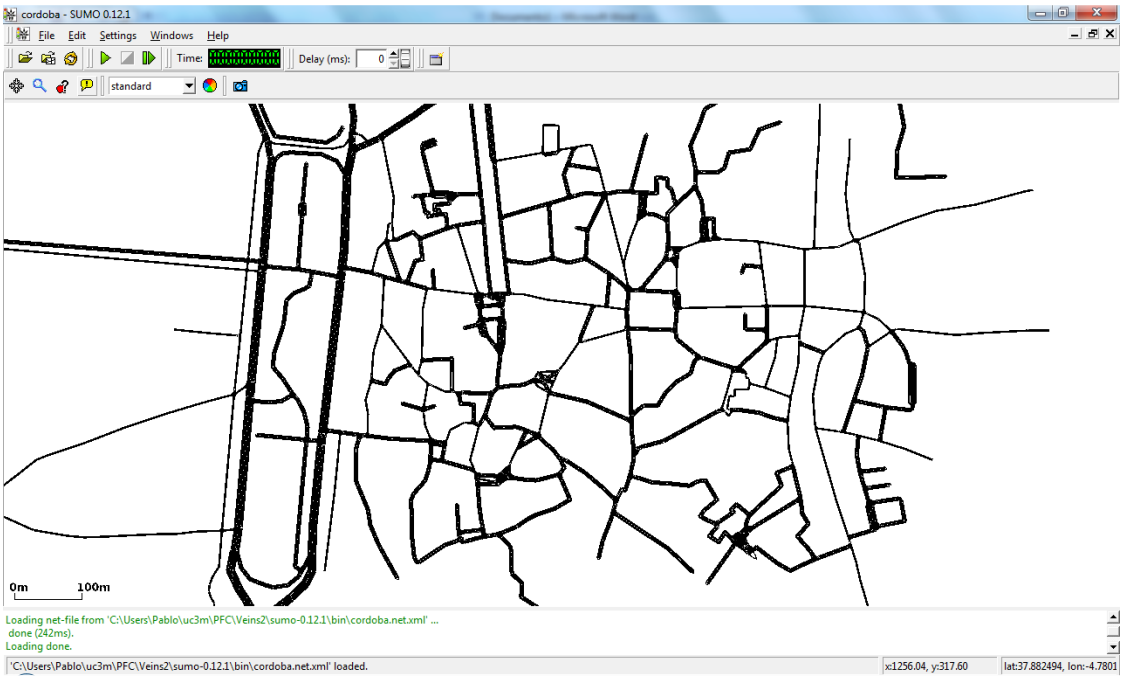


Figura 67. Visualización del mapa importado en SUMO tras la conversión

Una vez exportado el mapa de interés para nuestra simulación, y realizadas las modificaciones oportunas sobre el mismo, estamos en disposición de utilizarlo desde SUMO. Podemos ver el resultado en la Figura 67 anterior.

6.3.2.2 Generación de rutas

Teniendo definida la topología de la red, únicamente resta generar la denominada *demanda de tráfico*, es decir, la descripción de las rutas que seguirán los vehículos.

A partir de ahora se usará la siguiente nomenclatura: se denomina *viaje* al movimiento de un vehículo definido mediante el punto de partida y el de llegada. Por otra parte, se denomina *ruta* al conjunto total de enlaces por los que pasa el vehículo para realizar un *viaje*.

Hay varios métodos para generar la *demanda de tráfico* en SUMO, principalmente:

- Usando definiciones de rutas.
- Usando definiciones de viajes.
- Usando definiciones de flujos (similar al anterior, pero uniendo vehículos con similares viajes en grupos).
- Usando definiciones de flujos con porcentaje de giro en intersecciones (no se especifica el enlace de destino, y en su lugar se indica la probabilidad de realizar giros en intersecciones).
- Usando rutas aleatorias.

Nos centraremos en los primeros métodos, pues se trata de los más simples y útiles para crear rutas personalizadas. Finalmente se comentará cómo realizar la generación de rutas aleatorias. En cualquiera de los casos, se creará un fichero *.rou.xml* con la información deseada.

Para ello, lo primero que debemos decir a SUMO es qué tipo de vehículo deseamos simular.

Tipos de vehículo

La sintaxis para la definición de un tipo de vehículo es la siguiente (notar que esta definición deberá ir en el fichero de rutas *.rou.xml*, como se verá a continuación en más detalle):

```
<vtype id="tipo1" accel="0.8" decel="4.5" sigma="0.5" length="7.5"
maxspeed="70"/>
```

Los parámetros mostrados son los más típicos para el modelo de Krauß (de *coche seguidor*). Definen los siguientes atributos del vehículo (algunos opcionales):

- *id*: es el identificador del tipo de vehículo que se está definiendo (es un *string*).
- *accel*: es la aceleración en m/s^2 (debe ser de tipo *float*).
- *decel*: es la deceleración en m/s^2 (debe ser de tipo *float*).
- *sigma*: es un parámetro que da idea de la imperfección del conductor. Es un *float* entre 0 y 1.
- *lenght*: tamaño "bruto" del vehículo (vehículo + separación con el precedente). Se indica en metros mediante un *float*.
- *maxspeed*: velocidad máxima en m/s indicada mediante un *float*.
- *vclass*: clase abstracta del vehículo (opcional, ver abajo).

Existen algunos parámetros opcionales más, tales como color o forma de la visualización, para ello se remite al lector a [SDoc].

Clases abstractas de vehículos

SUMO introduce este concepto para dar la opción de permitir o bloquear el paso de determinados tipos de vehículos por determinados enlaces. Se trata de una funcionalidad bastante útil, por ejemplo, si se desea generar un carril para bus y taxi, que esté prohibido para el resto de vehículos.

Existen las siguientes clases abstractas principales:

- *"private"*
- *"public_transport"*
- *"public_emergency"*
- *"public_authority"*
- *"vip"*
- *"taxi"*
- *"bus"*
- *"delivery"*
- *"transport"*
- *"lightrail"*
- *"rail_slow"*

- *"rail_fast"*
- *"motorcycle"*
- *"bicycle"*
- *"pedestrian"*

a. Rutas

Una vez definido el tipo de vehículo, tenemos la opción de indicar la ruta específica para el mismo. Esto se debe hacer, igualmente, en el fichero *.rou.xml* (etiqueta *<routes>*).

```
<routes>
<vtype id="tipo1" accel="0.8" decel="4.5" sigma="0.5" length="5" maxspeed="70"/>

<vehicle id="0" type="tipo1" depart="0" color="1,0,0">
<route edges="calle_inicio calle_1 calle_2 calle_3 calle_destino calle_fin"/>
</vehicle>

</routes>
```

Como se puede observar, la ruta viene definida por un listado de enlaces por los que el vehículo pasa. En el ejemplo anterior, el vehículo circulará por los enlaces marcados por los identificadores *calle_inicio*, *calle_1*, *calle_2*, *calle_3* y *calle_destino* (por ese orden) y, tan pronto como se aproxime al enlace *calle_fin*, desaparecerá de la simulación.

Nótese que los enlaces que componen una ruta deben estar interconectados, pues SUMO no realiza esta comprobación.

SUMO también nos ofrece la posibilidad de definir rutas a seguir por más de un vehículo. Para ello, se debe utilizar una sintaxis como la siguiente, de forma que la ruta queda definida previamente, para más tarde incluir las distintas instancias de vehículo (indicando para cada una de ellas qué ruta seguirá).

```
<routes>
<vtype id="tipo1" accel="0.8" decel="4.5" sigma="0.5" length="5" maxspeed="70"/>
<route id="ruta0" color="1,1,0" edges=" c_inicio c_medio c_destino c_fin"/>

<vehicle id="0" type="tipo1" route="ruta0" depart="0" color="1,0,0"/>
<vehicle id="1" type="tipo1" route="ruta0" depart="0" color="0,1,0"/>

</routes>
```

Como vemos, la definición de cada vehículo incluye, además de su ruta, otros parámetros, tales como su tipo, identificador, etc. Veamos exactamente qué parámetros pueden definirse:

- *id*: *string* que identifica al vehículo.
- *type*: identificador del tipo de vehículo (*string*).
- *route*: nombre de la ruta que tomará el vehículo.
- *color*: color de la visualización.
- *depart*: instante (segundo) de la simulación en el que el vehículo entrará en la red (*float*).
- *departspeed*: velocidad con la que el vehículo entra en la red (*float ó 0,"random","max"*).

b. Rutas y distribuciones vehiculares

En lugar de definir las rutas y los tipos de vehículo de forma explícita para cada uno de ellos, SUMO puede escogerlas de forma aleatoria de entre una distribución dada.

```
<routes>

<vtypeDistribution id="dist_tipo1">
<vtype id="tipo1" accel="0.8" length="5" maxspeed="70" probability="0.9"/>
<vtype id="tipo2" accel="1.8" length="15" maxspeed="50" probability="0.1"/>
</vtypeDistribution>

<routeDistribution id="dist_ruta1">
<route id="ruta0" color="1,1,0" edges="c_1 c_2 c_3 c_4" probability="0.9"/>
<route id="ruta1" color="1,2,0" edges="c_1 c_2 c_3" probability="0.1"/>
</routeDistribution>

</routes>
```

Como se puede observar, en cada distribución debe haber varios tipos (de vehículo o de ruta) y cada uno debe llevar asociada una probabilidad.

A la hora de definir el vehículo, se debe proceder exactamente igual que en los casos anteriores, pero esta vez indicando como tipo de vehículo y de ruta, las distribuciones que se han creado:

```
<routes>
<vehicle id="0" type="dist_tipo1" route="dist_ruta1" depart="0" color="1,0,0"/>
</routes>
```

c. Viajes

Si en lugar de definir la ruta completa, se desea indicar únicamente cuál será el viaje que realice el vehículo, SUMO nos permite hacerlo. La sintaxis para ello es de la siguiente forma:

```
<tripdef id="viaje1" depart="0" from="calle_inicio" to="calle_fin"
type="v_tipo1" period="5" repno="3" color="0,0,1"/>
```

Todos los parámetros son bastante intuitivos, salvo *period*, que define el tiempo (en segundos) entre creación de vehículos, y *repno*, que indica el número de vehículos que se crearán cada vez.

d. Flujos

Si lo que se desea generar es un flujo de vehículos, en lugar de definir la ruta de cada uno de ellos por separado, se deberá utilizar la siguiente estructura sintáctica (definida en un fichero o como etiqueta interna):

```
<flows>
<flow id="flujo1" from="calle_inicio" to="calle_fin" begin="4" end="25" no="10"
type="v_tipo1" color="0,0,1"/>
</flows>
```

En este caso, el tiempo de salida toma un cariz diferente, ya que si un número (determinado por el parámetro *no*) de vehículos va a realizar la misma ruta, no pueden partir en el mismo instante. Por lo tanto, SUMO debe distribuir uniformemente el momento de salida de los vehículos entre los instantes determinados por *begin* y *end*.

e. Flujos con porcentaje de giro en las intersecciones

La aplicación JTRROUTER de SUMO permite al usuario definir flujos mediante una probabilidad de giro en intersecciones, es decir, la ruta dependerá del porcentaje asociado a cada giro en cada cruce. La llamada debe ser como sigue:

jtrrouter --flows=<FLUJOS> --turns=<GIROS> --net=<MI_RED>
--output-file=MisRutas.rou.xml -b <COMIENZO> -e <FIN>

La definición de los flujos se debe hacer como se ha indicado en el apartado anterior, mientras que la de los giros debe realizarse mediante un archivo con extensión *.turns.xml* que siga el siguiente formato. Como vemos, en el cruce se asigna una probabilidad determinada a cada una de las opciones de giro.

```
<turn-defs>
<interval begin="0" end="3600">
<fromedge id="calle0">
<toedge id="calle1" probability="0.2"/>
<toedge id="calle2" probability="0.7"/>
<toedge id="calle3" probability="0.1"/>
</fromedge>
</turn-defs>
```

f. Rutas aleatorias

Existe un *script* de Python desarrollado con el objetivo de producir rutas aleatorias, su nombre es *randomTrips.py*.

Actualmente, se trata del método más recomendable para lograr esta funcionalidad. No obstante, cabe notar que los resultados no siempre son del todo realistas.

La llamada a este *script* es de la siguiente forma:

randomTrips.py -n mi_red.net.xml -b 5 -e 50

Donde:

- -*n* indica la red para la cual se van a generar las rutas.
- -*b* es el instante inicial en el que comenzarán los vehículos a desplazarse (0 por defecto).
- -*e* indica el instante final

6.4 VEINS en funcionamiento

Para finalizar este capítulo, se presenta al lector un ejemplo práctico de VEINS en funcionamiento. Se mostrará paso a paso cada una de las fases para hacer que nuestra simulación se ejecute y se verá el resultado: SUMO y OMNeT++ trabajando en paralelo y bidireccionalmente acoplados.

Se trata de un ejemplo sencillo y visual, que pretende servir de guía para desarrollos más específicos. En concreto, se simulará una red con cuatro calles que se interceptan en un único cruce, gestionado mediante prioridad. La comunicación intervehicular se implementa mediante un protocolo IVC básico, consistente en que cada nodo reenviará por *broadcast* todos los mensajes que recibe.

En el ANEXO que se encuentra al final de esta memoria, se puede ver el código de todos los ficheros utilizados para esta simulación.

En primer lugar debemos generar nuestra red en SUMO. Para ello, generamos los ficheros de definición de nodos, de enlaces, de conexiones y de tipos (todos ellos se pueden ver en el ANEXO) que conforman la red descrita anteriormente. Posteriormente, llamamos a NETCONVERT tal y como se muestra en la captura de pantalla de la Figura 68 a continuación.

Figura 68. Generación de la red en SUMO mediante NETCONVERT

El siguiente paso es generar el fichero de demanda de tráfico, para posteriormente indicar a nuestro proyecto OMNeT++ que ése es el fichero de rutas que vamos a utilizar, así como que la red a simular es la que acabamos de generar.

Para ello, creamos un fichero *.sumo.cfg* con esta información (ver apartado 6.3.2), y además, indicamos estos parámetros en *sumo-launchd.launch.xml* presente en nuestro proyecto OMNeT++ (ver apartado 6.2.3.4). Podemos verlo en la siguiente Figura 69.

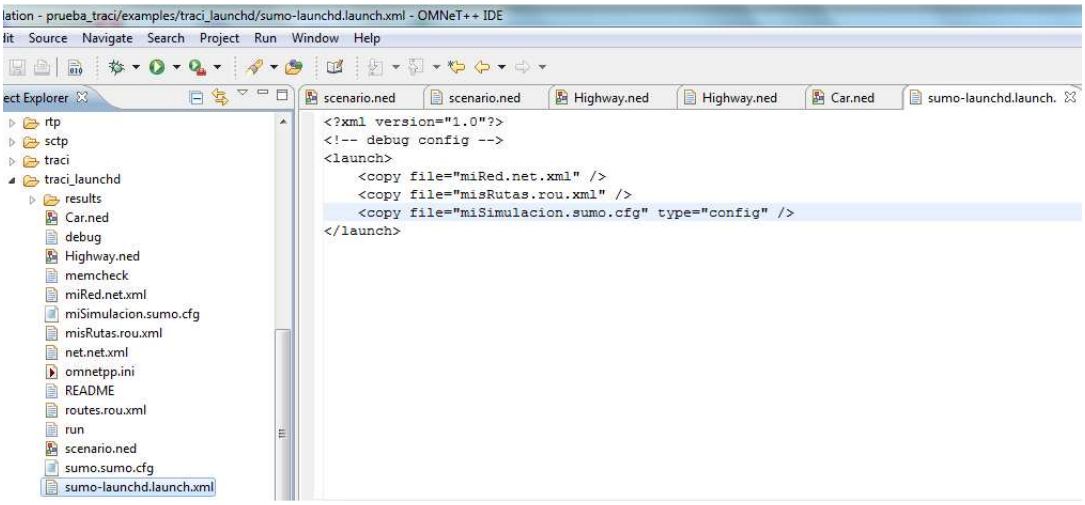


Figura 69. Parámetros en los ficheros de configuración del proyecto OMNeT++ que indican qué simulación de SUMO deseamos ejecutar

Como ya se ha mencionado, mediante la interfaz TraCI, se consigue llevar a cabo la interconexión entre SUMO y OMNeT++, la cual se realiza a través de una conexión cliente-servidor en TCP.

Para evitar al usuario tener que lanzar una instancia de SUMO por cada simulación de OMNeT++, este *script*, denominado *sumo-launchd.py* lanzará una nueva copia de SUMO cada vez que el simulador de red lo requiera.

De esta forma, *sumo-launchd* funciona como proxy entre ambos, tal y como se muestra en la siguiente Figura 70.

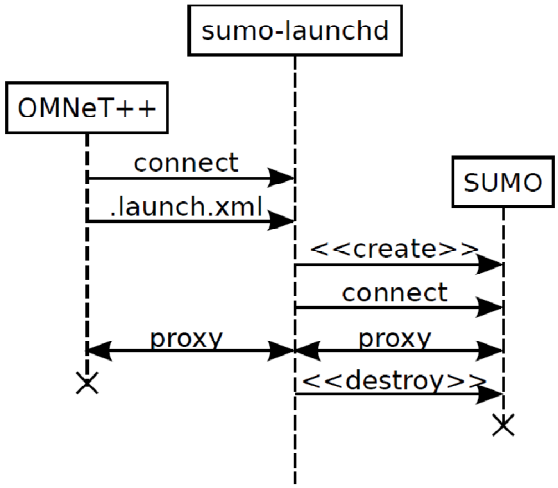


Figura 70. El script de Pyton sumo-launchd actuando como proxy entre OMNeT++ y SUMO

El siguiente paso, por tanto, será ejecutar el *script* de Python *sumo-launchd.py* que se encargará de lanzar las instancias de SUMO cada vez que OMNeT++ lo precise. Podemos ver su llamada e inicialización en la siguiente captura.

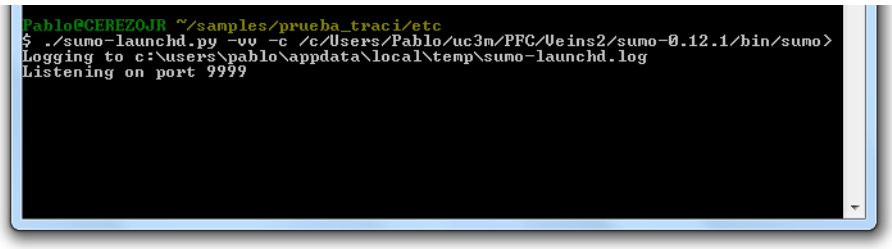


Figura 71. Ejecución e inicialización del script de Python sumo-launchd.py

En este punto, el sistema está listo para que lancemos la simulación desde OMNeT++.

Como vemos en la Figura 72 mostrada a continuación, la simulación se iniciará mediante ejecución del fichero *.ini* del proyecto OMNeT++: *Run as… OMNeT++ Simulation*.

Una vez iniciada la simulación, vemos que el *script* de Pyhton, que hace las veces de Proxy, establece la conexión entre ambos simuladores (ver Figura 73).

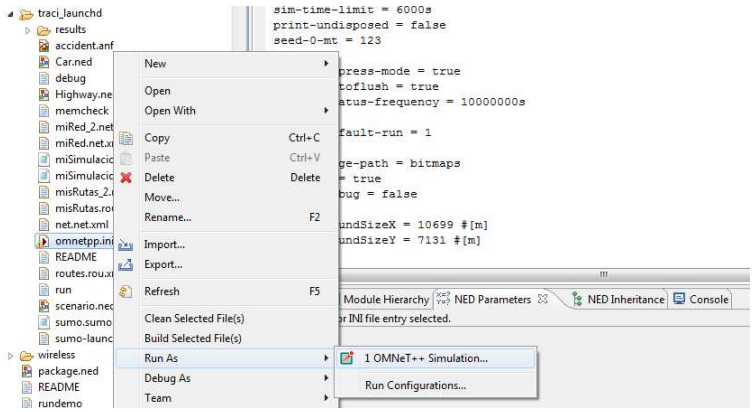


Figura 72. OMNeT++: inicio de la simulación

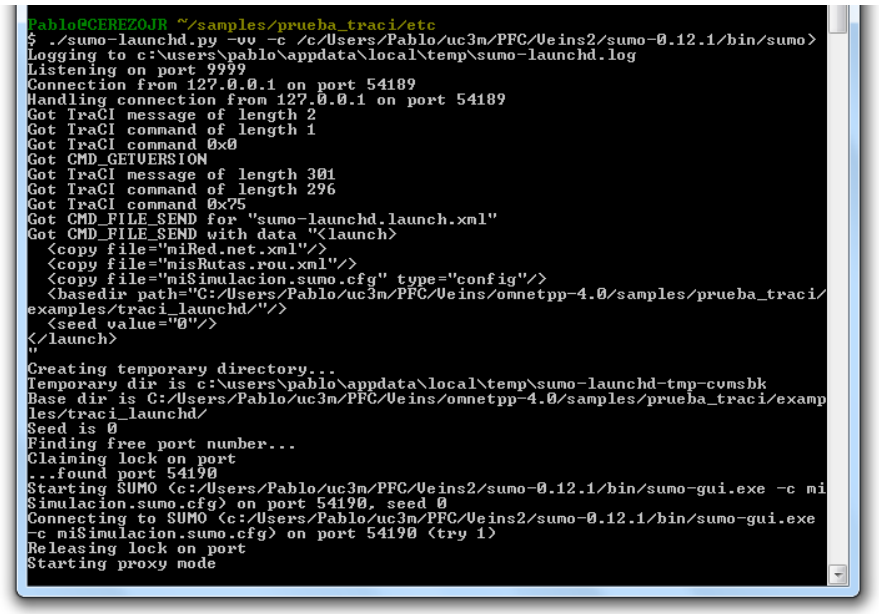


Figura 73. Inicialización del modo proxy: establecimiento de conexión entre SUMO y OMNeT++

En la Figura 74 siguiente se puede observar el log de eventos de OMNeT++. En el primer instante se inicializan los módulos *scenario*, *channelcontrol* y *TraCIScenarioManager*. Es decir, el módulo básico sobre el que se construyen todos los demás, así como los encargados de modelar el canal y la movilidad de los nodos.

A continuación, SUMO comenzará a producir vehículos de la forma indicada en el fichero de rutas, lo cual se comunicará a OMNeT++ mediante un mensaje TraCI (ver Figura 75: *TraCI message*).

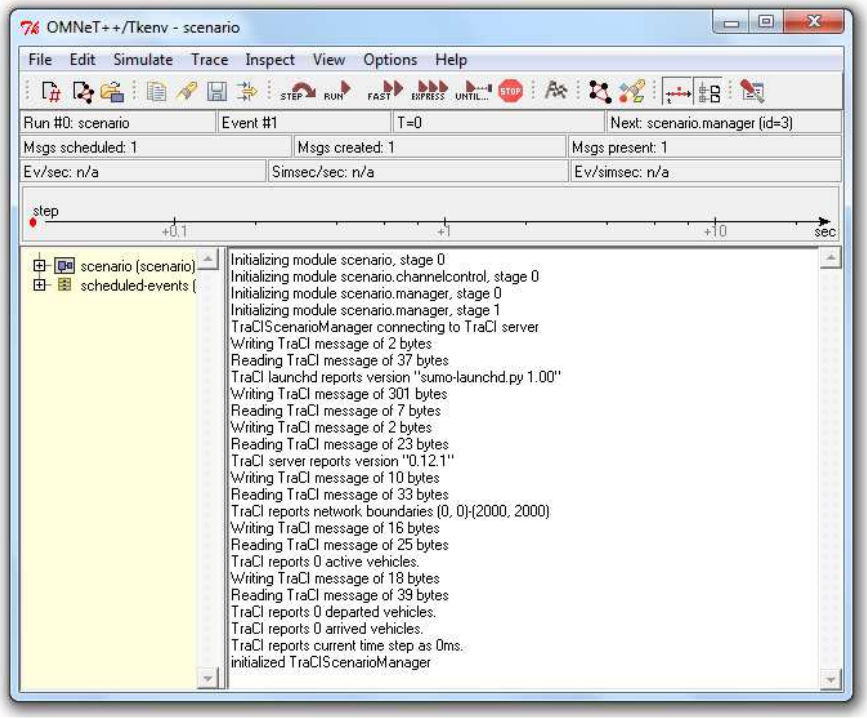


Figura 74. Instante 0 de la simulación: inicialización de los módulos scenario, channelcontrol y TraCIScenarioManager

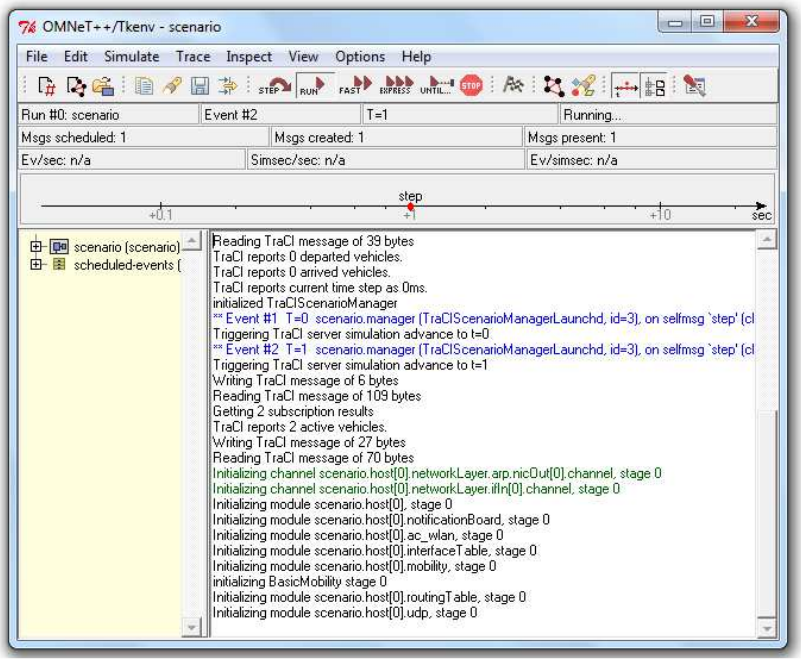


Figura 75. Intercambio de mensajes TraCI e inicialización de los módulos del primer vehículo (host)

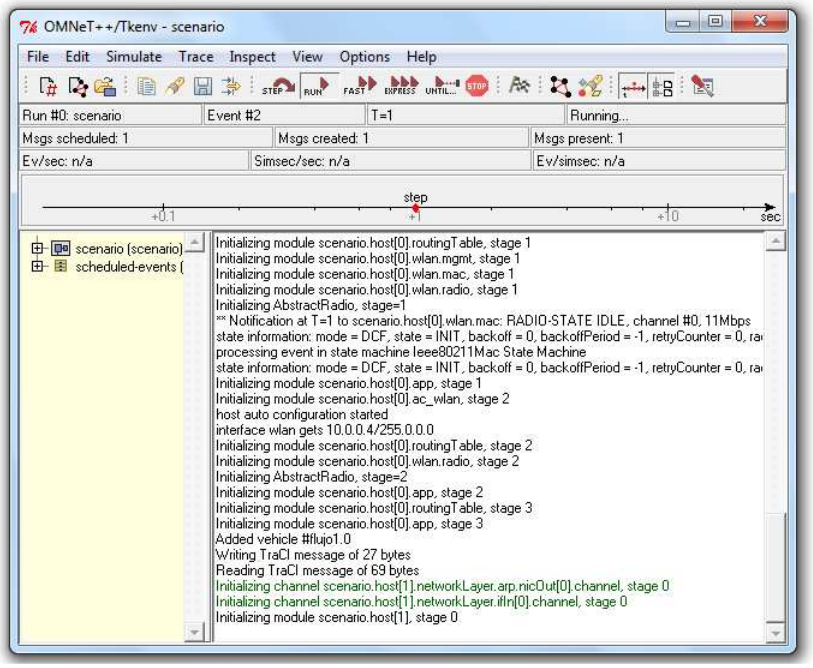


Figura 76. Un nuevo vehículo aparece en la simulación

A medida que van avanzando los instantes de la simulación, se van generando nuevos vehículos en SUMO, los cuales van apareciendo como nuevos nodos en OMNeT++. Esto se puede ver en el log de las capturas mostradas previamente.

No obstante, de forma más visual, podemos apreciarlo en las interfaces gráficas de ambos simuladores. En la Figura 77 mostrada a continuación, podemos ver la ventana de OMNeT++ a la izquierda y la de SUMO a la derecha.

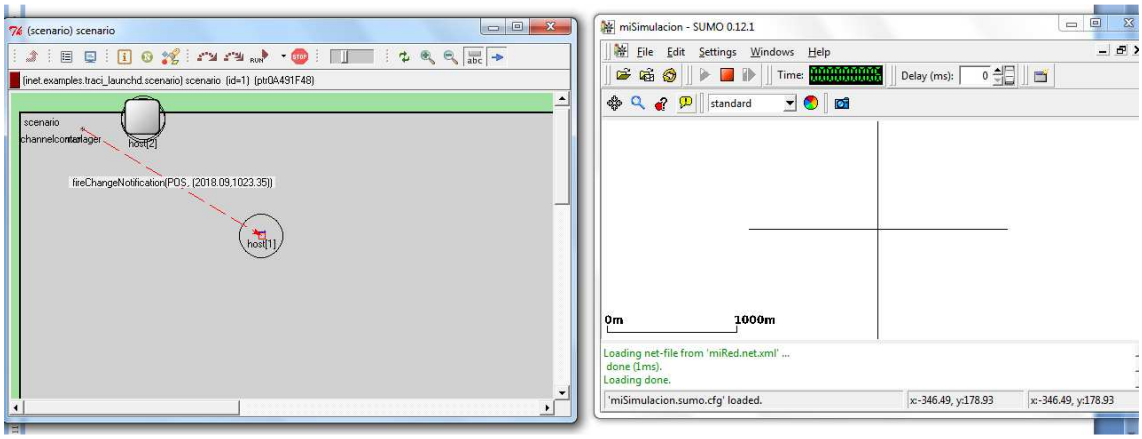


Figura 77. Interfaces gráficas de ambos simuladores (izquierda: OMNeT++, derecha: SUMO) durante la ejecución de la simulación

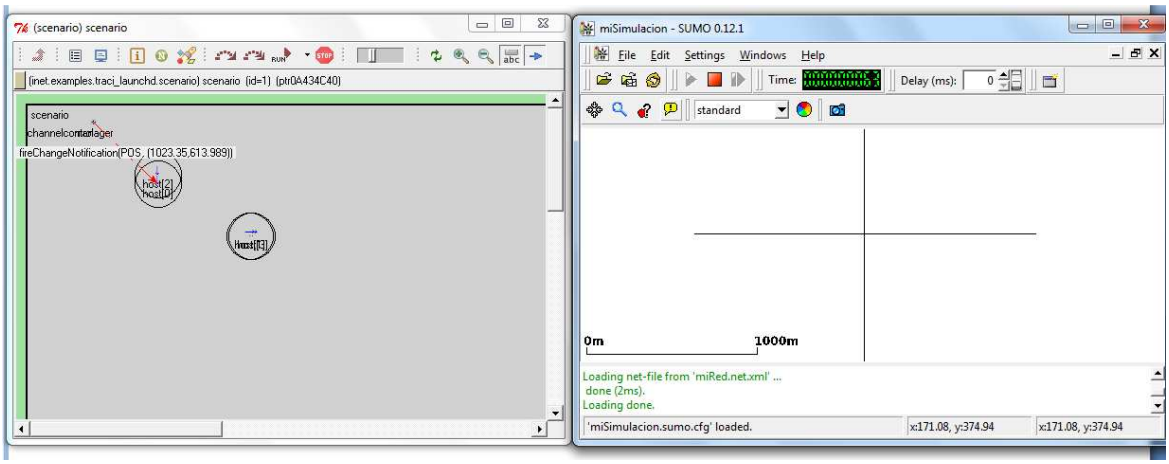


Figura 78. Interfaces gráficas de ambos simuladores en un instante posterior de la simulación

Como se observa en las figuras anteriores, OMNeT++ está generando nodos tal y como le indica SUMO. Vemos que aparecen vehículos desde el punto norte y el este y se van desplazando según las condiciones indicadas en el fichero de demanda de tráfico.

En las capturas mostradas a continuación podemos ver cómo OMNeT++ conoce, mediante mensajes TraCI, las siguientes posiciones de los flujos que se están desplazando en SUMO. Es el módulo *TraCIScenarioManager* el encargado de comunicar las nuevas posiciones a cada uno de los nodos.

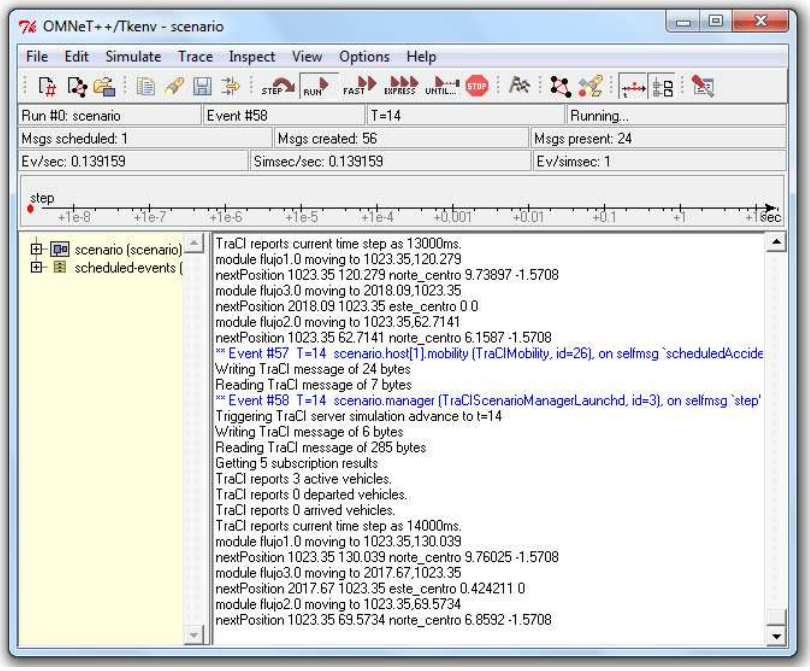


Figura 79. Envío de mensaje con la nueva posición de un nodo

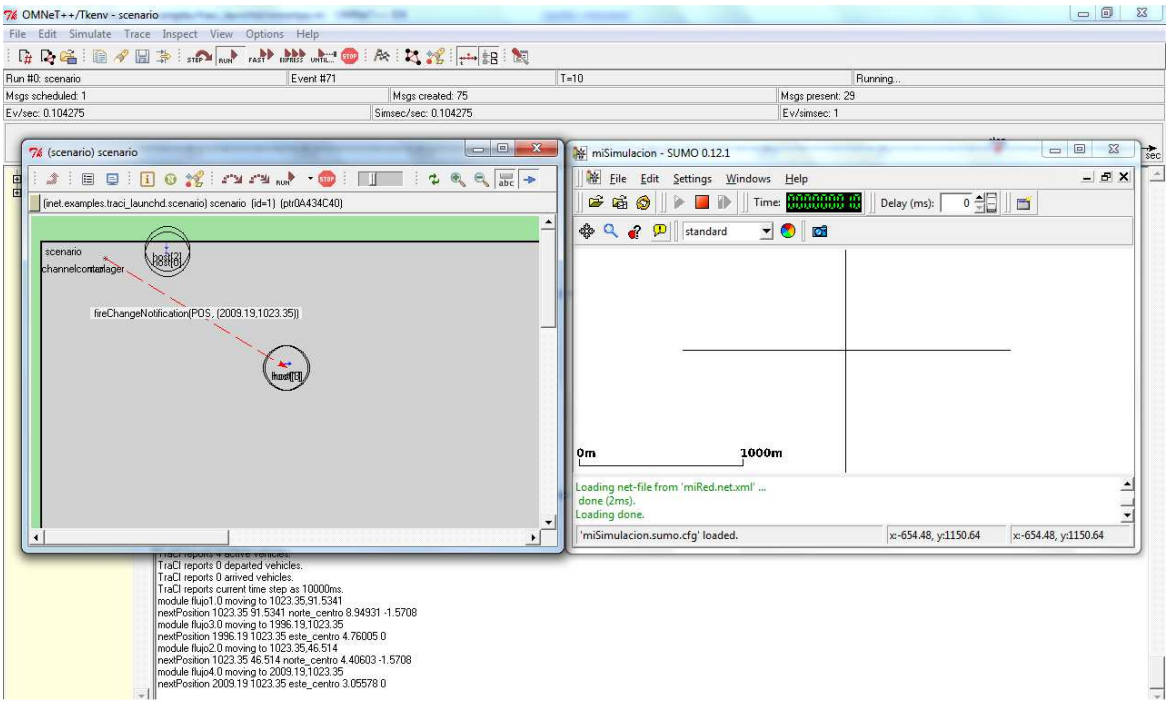


Figura 80. El nodo la notificación de su nueva posición (acorde con el movimiento en SUMO) y se desplaza

Para finalizar, se muestra una captura similar a las anteriores, pero con mayor aumento en la visualización de la red de SUMO. Se puede observar mucho mejor cómo los vehículos se están desplazando de la misma forma en ambos simuladores.

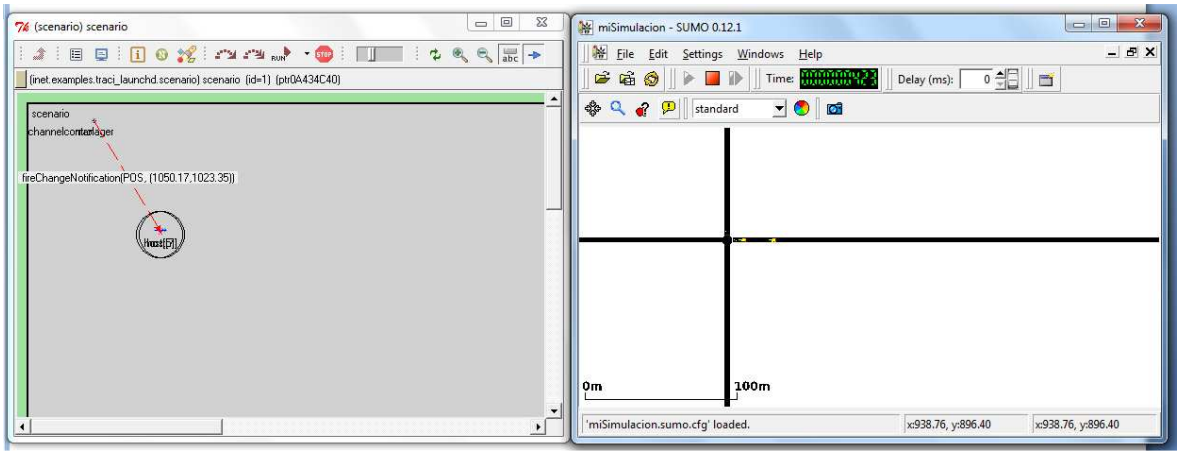


Figura 81. Visualización con zoom para SUMO, donde se observan los vehículos desplazándose

Análisis de los resultados de la simulación

El análisis de los resultados de la simulación es un proceso complejo y que dependerá de cuál sea la variable que se desee estudiar en concreto. OMNeT++ almacena

los resultados como valores escalares, vectoriales y como histogramas. Como se puede observar, en el proyecto aparecerá un carpeta denominada *results* que los almacena (escalares con extensión *.sca* y vectores con extensión *.vec*). A partir de ellos, el usuario puede aplicar métodos estadísticos para extraer la información que le resulte relevante según sus objetivos.

OMNeT++ cuenta con una herramienta de análisis estadístico integrada en su IDE. Las preferencias del usuario, es decir, qué es lo que desea obtener de los datos sin procesar, deben ser indicadas en un fichero de análisis de OMNeT++, el cual, como veremos, tiene extensión *.anf*. De esta forma, si se vuelve a ejecutar una simulación, no es preciso volver a realizar, manualmente, el análisis deseado. Simplemente indicando al simulador cuál es el fichero de resultados, los nuevos diagramas o gráficos se generarán con las nuevas trazas.

Al crear el fichero de análisis, el usuario debe indicar a qué conjuntos de resultados está asociado (es decir, qué ficheros *.vec* y/o *.sca*). Para ello, en el menú se debe elegir *File* > *New* > *Analysis File*, tal y como se indica en la Figura 82.

Veamos qué aspecto tiene trabajando en el marco de simulación VEINS. Como podemos ver, en el proyecto creado, el fichero de análisis se denomina *accident.anf*, pues está asociado a los resultados de la simulación correspondiente al fichero *.ini* del que se habló en el apartado 6.2.3.4.

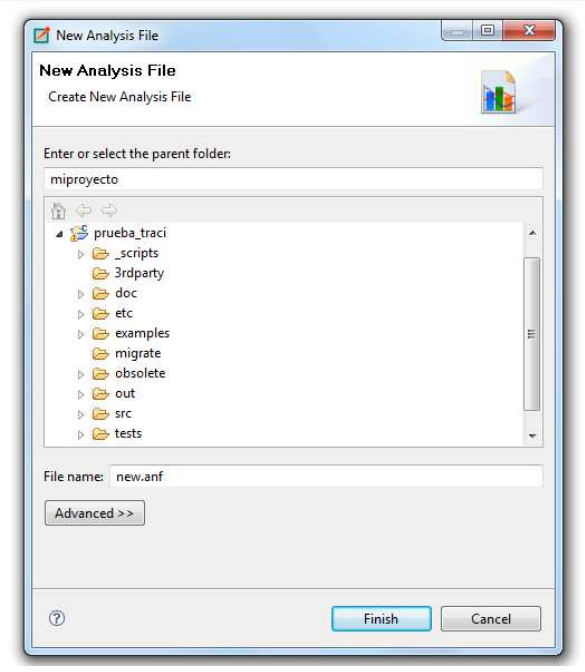


Figura 82. Creación de un fichero de análisis (.anf) en OMNeT++

En la siguiente Figura 83, vemos que en la sección superior aparecen los ficheros de entrada para nuestro análisis, tanto los escalares como los vectoriales. Estos ficheros se pueden incluir mediante el diálogo que surge al hacer clic en el botón *Add File…*

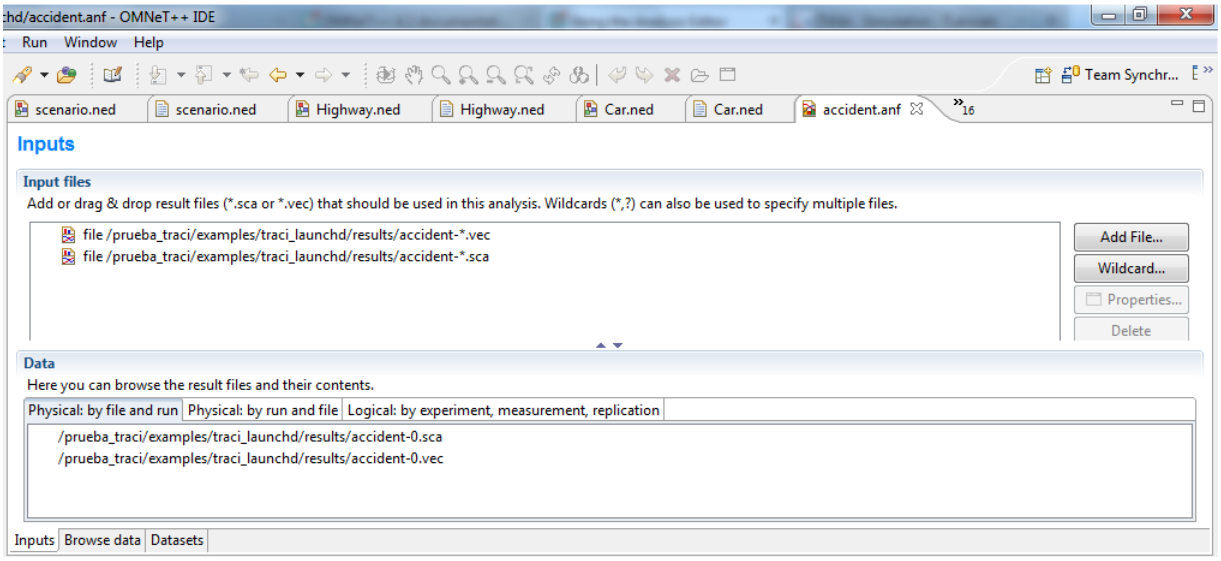


Figura 83. Pestaña Inputs de nuestro fichero de análisis (accident.anf)

Como se observa en la figura anterior, la siguiente pestaña a los ficheros de entrada (pestaña *Inputs*) es la denominada *Browse Data*. Su finalidad es mostrar al usuario los datos concretos que se han cargado de los ficheros de resultados, tanto escalares como vectoriales. Desde este menú pueden observarse todos los datos que han sido generados durante la simulación, así como aplicar filtros sobre los mismos.

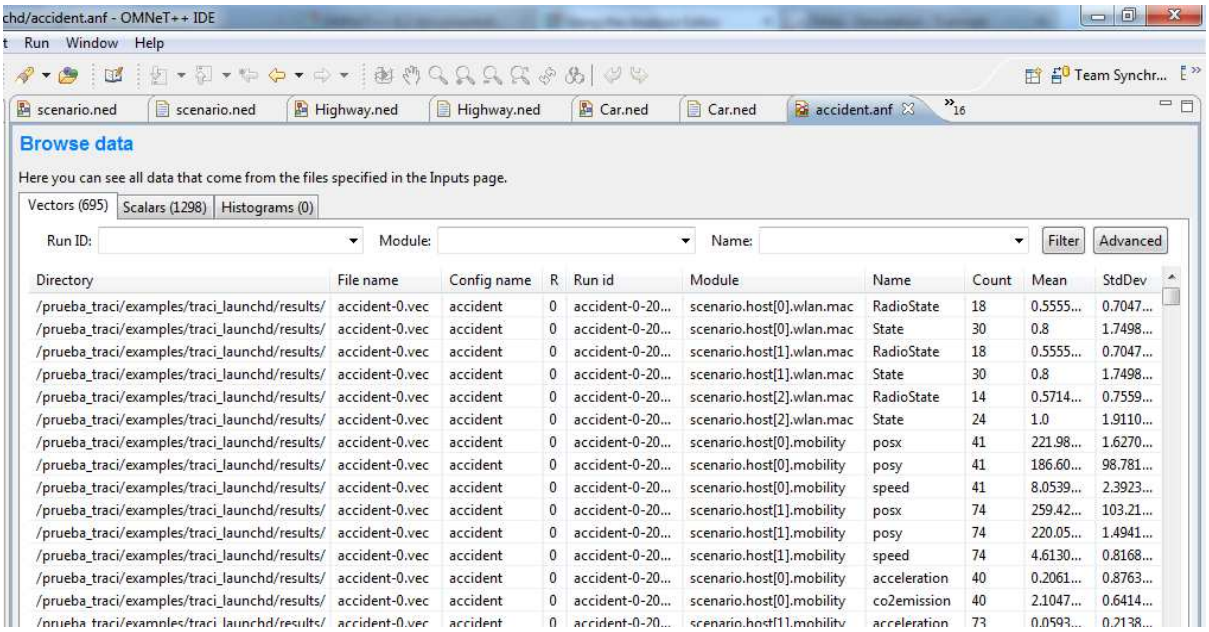


Figura 84. Pestaña Browse Data, donde se observan los datos vectoriales

Desde este mismo punto, OMNeT++ permite mostrar en forma de gráfico la información que se desee, ya sea habiendo aplicado algún filtro o para el conjunto de los datos. No obstante, se ofrece al usuario una tercera pestaña denominada *Datasets*, la cual sirve para realizar el procesado de los datos de una manera más automática.

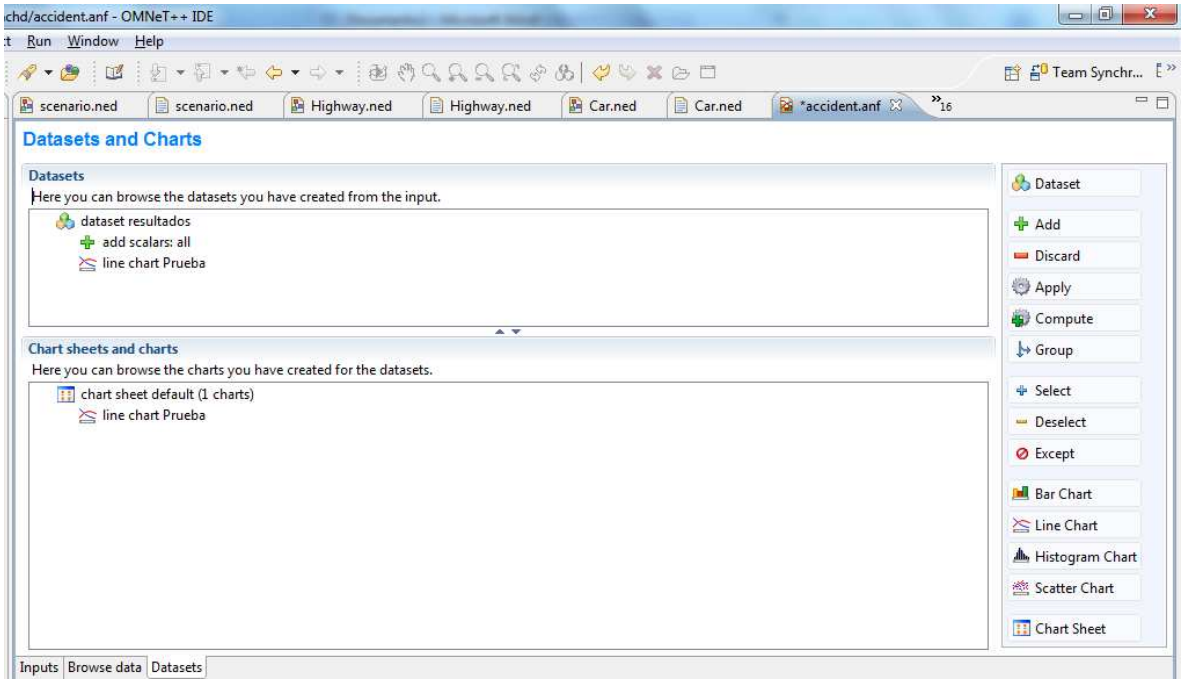


Figura 85. Pestaña Datasets: se incluye una representación gráfica de todos los datos escalares

Desde esta pestaña, el usuario puede describir qué conjunto de datos desea procesar, indicar cuál es el procesado que quiere aplicar y en qué tipo de diagrama hacerlo. Existe un menú con el que añadir datos, descartarlos, filtrarlos, elegir las operaciones y el contenido de los diagramas y finalmente, para crearlos. Podemos observar lo descrito en la Figura 85 mostrada anteriormente.

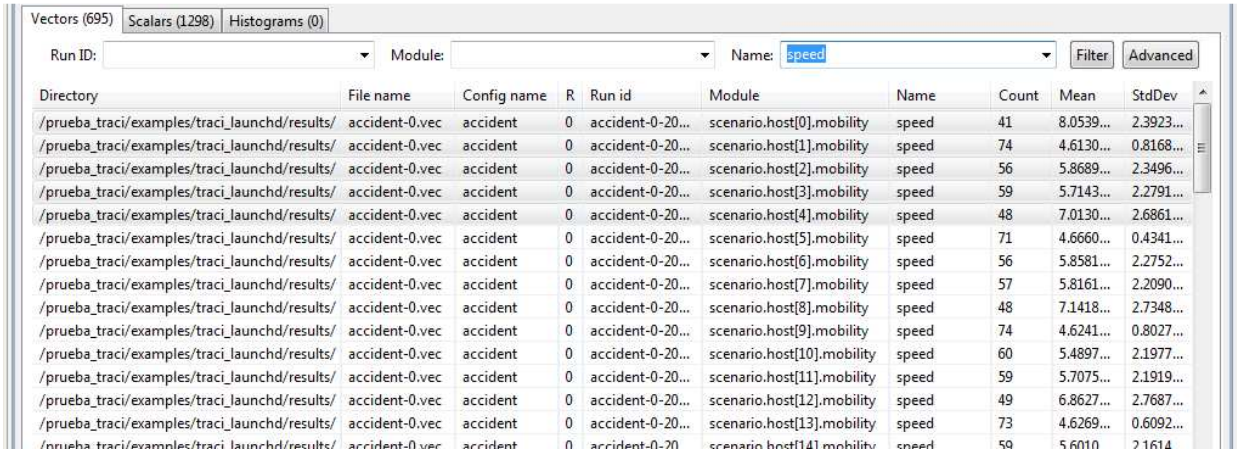
Finalmente, cabe mencionar que OMNeT++ ofrece la posibilidad de exportar el contenido de los mencionados conjuntos de datos en ficheros de texto. Existen tres formatos soportados:

- CSV (Comma Separated Values).
- Ficheros de texto Octave.
- Script de Matlab.

Para realizar la exportación, se debe realizar clic con el botón derecho en el diagrama deseado, para posteriormente seleccionar el submenú *Export File*.

Como se ha dicho, el proceso que se desee seguir (qué datos procesar, qué procesado aplicar, cómo representarlos, con qué sistema exportarlos en caso de ser necesario, etc.) dependerá de cuáles sean las variables bajo estudio. No obstante, a modo de ejemplo se presenta a continuación la representación gráfica de un resultado bastante característico de la simulación ejecutada anteriormente.

Podemos ver en la Figura 86 cómo se ha realizado un filtrado en los datos vectoriales, de tal forma que se muestran únicamente las velocidades de los nodos (*speed*). Seleccionando, por ejemplo, los cinco primeros y haciendo clic, podemos verlos representados.



Directory	File name	Config name	R	Run id	Module	Name	Count	Mean	StdDev
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[0].mobility	speed	41	8.0539...	2.3923...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[1].mobility	speed	74	4.6130...	0.8168...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[2].mobility	speed	56	5.8689...	2.3496...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[3].mobility	speed	59	5.7143...	2.2791...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[4].mobility	speed	48	7.0130...	2.6861...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[5].mobility	speed	71	4.6660...	0.4341...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[6].mobility	speed	56	5.8581...	2.2752...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[7].mobility	speed	57	5.8161...	2.2090...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[8].mobility	speed	48	7.1418...	2.7348...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[9].mobility	speed	74	4.6241...	0.8027...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[10].mobility	speed	60	5.4897...	2.1977...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[11].mobility	speed	59	5.7075...	2.1919...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[12].mobility	speed	49	6.8627...	2.7687...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[13].mobility	speed	73	4.6269...	0.6092...
/prueba_traci/examples/traci_launchd/results/	accident-0.vec	accident	0	accident-0-20...	scenario.host[14].mobility	speed	59	5.6010...	2.1614...

Figura 86. Datos vectoriales con filtrado aplicado para obtener la velocidad

En la siguiente Figura 87 tenemos la representación gráfica de los datos de ejemplo. Como se puede ver, se muestra la velocidad (en m/s) frente al instante de simulación (en segundos). Cada color está asociado a uno de los cinco nodos (vehículos) para los que se están mostrando los datos, tal y como indica la leyenda superior.

Se puede observar que los vehículos 0 (azul), 1 (rojo) y 2 (verde oscuro) salen en primer lugar, mientras que el 3 (amarillo) y el 4 (verde claro) lo hacen más avanzada la simulación. Para todos ellos observamos un comportamiento bastante lógico: aceleran hasta alcanzar una velocidad constante (asociada a la permitida en la vía) para finalmente abandonar la simulación (cuando han llegado a su destino). En algunos casos, como se puede observar, los vehículos han realizado una parada (con la consiguiente deceleración previa), la cual es debida a que han alcanzado la posición del cruce y se han encontrado con otros vehículos presentes en él.

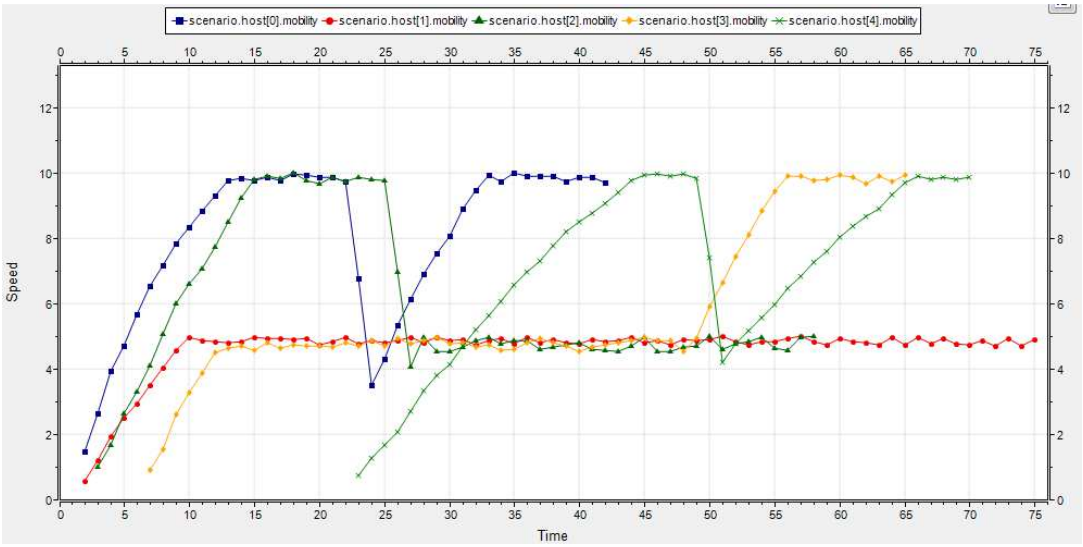


Figura 87. Representación de la velocidad para 5 vehículos a lo largo de la simulación

6.5 Resumen

El capítulo que ahora se cierra, puede considerarse la columna vertebral de esta memoria, pues se han presentado las guías para la creación y ejecución de las simulaciones vehiculares, lo cual era el objetivo fundamental.

Se ha visto cómo crear redes en SUMO y en OMNeT++, así como una parte muy importante de sus funcionalidades extras. El lector puede considerarse autónomo para la creación de entornos de simulación tras el estudio de esta guía.

Finalmente, se ha mostrado un ejemplo práctico de VEINS en funcionamiento. En él se indican, paso a paso, las etapas de un proceso de simulación y se termina demostrando que gracias al sistema TraCI, ambos simuladores se ejecutan en paralelo y bidireccionalmente acoplados.

Capítulo 7

Conclusiones y trabajo futuro

7.1 Conclusiones

En el transcurso del trabajo realizado durante este Proyecto Fin de Carrera, que ha quedado plasmado a lo largo de estos seis capítulos, se ha llevado a cabo una incursión en profundidad en el campo de las redes vehiculares. Se ha dado a conocer el concepto de VANET y su enorme potencial: la amplia gama de aplicaciones y las ventajas que aportan de cara a la seguridad vial, principalmente.

Es claro que el futuro -por no decir el presente- del campo de la automovilística y su seguridad, de los sistemas de asistencia a la conducción y de otros servicios de entretenimiento y control vial, pasa por la implementación de sistemas de comunicación intervehicular que funcionen sobre este tipo de redes.

Pero su completo desarrollo se ha visto ligado a una serie de factores y requerimientos tecnológicos, para los cuales ya se han dado pasos muy importantes, tanto en lo que se refiere al compromiso de entidades nacionales e internacionales, como a la consecución de estándares para comunicaciones inalámbricas asociados a este fin concreto. En este sentido, la comunidad científica e investigadora ha realizado

importantes avances que se han traducido en consorcios y sistemas reales de comunicación entre vehículos y con elementos de la infraestructura de las carreteras.

No obstante, pese a que todo esto juega un papel fundamental en la expansión de este tipo de sistemas y en su adopción por parte de todos los actores implicados, de cara al desarrollo de los denominados protocolos de comunicación intervehicular, es de vital importancia que la comunidad investigadora cuente con sistemas fiables de simulación.

La simulación tiene un rol absolutamente principal de cara a hacer que el mundo de las redes vehiculares dé el salto definitivo hacia su completo desarrollo e implementación. Muchos proyectos VANET dependen de forma totalmente directa de estos sistemas, para lograr ver la luz. Y es que gracias a estas herramientas de modelado de la realidad, se pueden obtener resultados fiables sin necesidad de recurrir a experimentos de campo, lo que permite llegar a nuevas soluciones sin la implicación de inversiones a ciegas, ni de construcción de costosas infraestructuras.

Siguiendo este enfoque, son multitud los proyectos que se han desarrollado en los últimos años de cara a crear marcos de simulación realistas diseñados expresamente para este tipo de redes. Para ello, necesariamente se precisa de un módulo para el modelado de la red (comunicación intervehicular propiamente dicha) y otro para la simulación de la movilidad (tráfico rodado). En el estado del arte se pueden encontrar diferentes soluciones para aunar ambos mundos, no obstante, el modelo más acertado, por sus prestaciones, es el denominado *simulador híbrido*.

Los simuladores híbridos de VANETs se caracterizan por el uso de un simulador de red y otro de tráfico con buenas prestaciones (demostrables cuando están trabajando de forma independiente), unidos mediante un módulo de comunicación entre ambos. Este módulo será el encargado de transmitir la información relativa a la movilidad (proveniente del simulador de tráfico) al simulador de red, y de igual modo, hacer que los mensajes que se intercambian los vehículos (simulador de red) lleguen al simulador de tráfico, de tal forma que la movilidad se vea influenciada por ellos.

En este contexto surge el sistema de simulación VEINS (*Vehicles in Network Simulation*), el cual cabe ser considerado como el simulador híbrido de VANETs con mayor progresión y futuro, dentro de los sistemas de código abierto disponibles entre la comunidad investigadora.

VEINS está basado en el simulador de redes OMNeT++, el simulador de tráfico SUMO y un sistema de comunicación entre ambos en tiempo real, denominado TraCI.

Tanto SUMO como OMNeT++ disponen de muy sofisticados modelos de movilidad y de comunicaciones (implementación de los estándares de interés y simulación del canal

inalámbrico), respectivamente, que hacen que aplicados al mundo vehicular, sus resultados sean realmente buenos.

Pero la gran aportación de VEINS es el denominado acoplamiento bidireccional entre estos dos simuladores, el cual ofrece grandes ventajas, de cara al realismo de los resultados, sobre otras soluciones desacopladas. Esto queda bien resumido en la siguiente declaración:

Si los resultados del módulo de simulación de red pueden afectar a cómo se desarrollará el movimiento de los vehículos, entonces se precisa de un sistema de interacción en tiempo real entre la simulación de movilidad y el módulo de simulación de red, ya que una metodología offline y aislada no es capaz de generar un resultado del todo realista.

VEINS, y en concreto el módulo TraCI, trabajan para solventar este conflicto, logrando así una interfaz de control muy minucioso entre los dos mundos.

Este tipo de herramientas de código abierto evolucionan de una forma relativamente rápida, pues muy diversos equipos y desarrolladores individuales trabajan en la implementación de mejoras y nuevas funcionalidades, por lo que las indicaciones mostradas en esta memoria referentes a las versiones de los distintos módulos, pueden perder valor en un futuro próximo. No obstante, tras las múltiples pruebas y problemas de incompatibilidad detectados durante el desarrollo de esta memoria, se ha podido concluir que a fecha de su publicación, se debe recomendar al usuario utilizar la versión 0.12.1 de SUMO y la versión 4.0 de OMNeT++.

Mediante los pasos de instalación y uso descritos, se lograrán los resultados realistas deseados para esta conjunción de versiones, y podremos afirmar que el desarrollador de sistemas de comunicación intervehicular contará con una herramienta fiable y de una enorme utilidad para su trabajo y, por consiguiente, para el avance de las VANETs en su conjunto.

Por lo tanto, se puede decir que el trabajo aquí realizado supone una herramienta de enorme utilidad sobre la que sustentar la labor de la comunidad desarrolladora. Se ha llevado a cabo una tarea de investigación y análisis de los problemas existentes y requerimientos de cara al crecimiento y expansión de las VANETs y se ha concluido que la simulación en este ámbito es fundamental.

En este sentido, los esfuerzos han ido destinados a arrojar luz sobre las herramientas de las que se disponen, y en mucha mayor profundidad, sobre VEINS.

Este exhaustivo estudio, las conclusiones obtenidas y los pasos para la puesta en marcha y uso de VEINS, que han sido detalladamente descritos, suponen una muy

completa guía que se ofrece a la comunidad científica e investigadora en general y a la desarrolladora en particular.

En definitiva, la labor aquí desempeñada, que ha quedado reflejada en forma de esta memoria, aporta un pilar más en el trabajo hacia el crecimiento de las redes vehiculares, particularmente desde la óptica del desarrollador de sistemas de comunicación intervehicular. Gracias a este proyecto, la comunidad investigadora cuenta con una extensa guía teórica y práctica sobre la que basar sus esfuerzos.

7.2 Trabajo futuro

Como cierre al trabajo desarrollado durante este Proyecto Fin de Carrera, se exponen a continuación algunas líneas de trabajo futuro, partiendo del aquí desempeñado, que han sido identificadas:

- **Estudio en profundidad de otras soluciones de simulación VANET de código abierto**
  Entre las que cabe destacar a TraNS (SUMO y ns-2), pues se trata de otro sistema de simulación híbrido con una gran aceptación entre la comunidad investigadora. Sería de un enorme interés comparar resultados prácticos de simulaciones con VEINS y con TraNS, de forma que se realizara un análisis exhaustivo de los puntos fuertes y débiles de cada uno de ellos con respecto al otro. Esto ayudaría, posiblemente, al desarrollo de ambos hacia mejoras en sus funcionalidades.

- **Trabajos enfocados al desarrollo de un sistema unificado de instalación para el marco VEINS**
  Se han detectado numerosas dificultades en la fase de instalación del sistema completo, lo que ha llevado a la necesidad de una inversión temporal considerable, previa al trabajo con VEINS como tal. Esto hace pensar que sería muy conveniente definir un proyecto dirigido a realizar un sistema de instalación mucho más automático y único, tanto para plataforma Windows, como Linux, que englobara a OMNeT++, SUMO, los módulos propios de VEINS y el resto de software auxiliar del que se ha hecho mención en el capítulo 5.

- **Desarrollo o mejora de los modelos para la simulación del canal inalámbrico**

  Como se ha visto, en VEINS se opta por una simulación sencilla del canal inalámbrico, mediante un modelo basado en datos experimentales para un entorno urbano. No obstante, sería interesante trabajar en el desarrollo de algunas mejoras enfocadas a contemplar una más amplia gama de escenarios y que nos permitiera, por ejemplo, establecer la presencia de objetos concretos en puntos concretos que impidan la correcta transmisión de la señal.

- **Implementación de sistema para realizar baterías de simulaciones con VEINS**

  El principio de funcionamiento de VEINS hace que sea preciso crear nuevas instancias de SUMO cada vez que OMNeT++ así lo requiere. Para facilitar este proceso, como se ha visto, se dispone del *script* de Python *sumo-launchd.py*, pero esta solución aún no contempla un escenario en el que el usuario desee lanzar baterías de simulaciones. Sería muy interesante trabajar en la dirección del desarrollo de un nuevo sistema que permitiera este mismo proceso, pero a la misma vez enfocado a poder lanzar múltiples simulaciones con una sola llamada.

- **Análisis pormenorizado de los resultados de la simulación en VEINS para escenarios más complejos**

  Aquí se han sentado las bases teóricas, referentes a los paradigmas básicos de funcionamiento, así como en lo relacionado con la instalación y puesta en marcha de este sistema de simulación, pero sólo se han mostrado resultados para escenarios básicos. La simulación de una topología de red más compleja, como podría ser la de una pequeña ciudad, podría arrojar más claridad de cara al trabajo con VEINS y algunas de sus funcionalidades más avanzadas.

Glosario

ABS	*Antilock Brake System*
ARP	*Address Resolution Protocol*
ATM	*Asynchronous Transfer Mode*
DSRC	*Dedicated Short Range Communications*
GNSS	*Global Navigation Satellite System*
GPS	*Global Positioning System*
ICMP	*Internet Control Message Protocol*
IDE	*Integrated Developing Environment*
IEEE	*Institute of Electrical and Electronics Engineers*
IGMP	*Internet Group Management Protocol*
IPv4	*Internet Protocol version 4*
IPv6	*Internet Protocol version 6*
ITS	*Intelligent Trasnportation System*
IVC	*Inter-Vehicular Communication*
JRE	*Java Runtime Environment*
LAN	*Local Area Network*
MAC	*Media Access Control*
MANET	*Mobile Ad-hoc Network*
MPLS	*Multiprotocol Label Switching*
NED	*Network Description*
OSPF	*Open Shortest Path First*
RSU	*Road Side Unit*
SCTP	*Stream Control Transmission Protocol*
TIS	*Traffic Information System*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
VANET	*Vehicular Ad-hoc Network*
VEINS	*Vehicles in Network Simulation*
WAN	*Wide Area Network*
XML	*Extensible Markup Language*

Referencias

[Arb10] Hadi Arbabi, Michele C. Weigle. *Highway Mobility and Vehicular Ad-hoc Networks in NS-3*. Vol. Proceedings of the 2010 Winter Simulation Conference. 2010

[ASL] *Active State Dynamic Language Experts*. Disponible [Internet]: <http://www.activestate.com> [julio de 2011]

[C2C] *Car-2-Car Comminication Consortium (C2C-CC)*. Disponible [Internet]: <http://www.car-to-car.org> [enero de 2012]

[CLIFF] *Connected Cruise Control Project*. CLIFFORD Electronics Benelux B.V. Disponible [Internet]: <http://www.clifford.nl/nl/projects-ccc.html> [agosto 2011]

[CVIS] *Cooperative Vehicle-Infrastructure Systems*. European Commission, Information Society and Media. Disponible [Internet]: <http://www.cvisproject.org > [julio de 2011]

[DOT] *Vehicle safety applications*. US Department of Transportation IntelliDrive Project – ITS Joint Program Office. 2008.

[DSRC] *"What is DSRC?"*. Standard Programs Lee Armstrong. Disponible [Internet] <http://www.leearmstrong.com/DSRC/DSRCHomeset.htm> [mayo 2012]

[ETSI] *Intelligent Transportation System (ITS): Vehicular Communications*. ETSI ITS Specification TR 102 638. 2009

[EUC] *Cars that talk: Commission earmarks single radio frequency for road safety and traffic management*. European Commission. 2008-08-05

[Fer11] José Luis Ferrás Pereira. *An Integrated Architecture for Autonomous Vehicles Simulation*. Proyecto Fin de Máster, Facultad de Ingeniería de la Universidad de Oporto. Noviembre 2011.

[Fio08] M. Fiore. *Vehicular mobility and Network Simulation.* Handbook on Vehicular Networks. 2008

[Gaw98] Christian Gawron. *Simulation-based Traffic Assignment*. Disertación inaugural. 1998

[GTNETS] Modeling & Analysis of Networks via Computer Simulations (MANIACS). *Georgia Tech Network Simulator (GTNetS).* Disponible [Internet]: <http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/ > [diciembre de 2011]

[Han08] Hannes Hartenstein. A Tutorial Survey on Vehicular Ad-hoc Networks. Universidad de Karlsruhe, Kenneth P. Laberteaux, Toyota Technical Center. IEEE Communications Magazine. Junio 2008

[Har09] Jérôme Härri, Fethi Filali and Christian Bonnet. *Mobility models for vehicular Ad -hoc networks: A survey and taxonomy*. IEEE Communications surveys & tutorials, Vol. 11, No. 4. 2009

[INET] *INET Framework for OMNeT++ Manual*. INET Framework Community. Disponible [Internet]: <http://www.inet.omnetpp.org> [noviembre de 2011]

[ITS] *IEEE Intelligent Transportation Systems Society*. Disponible [Internet]:<http://ewh.ieee.org/tc/its > [julio de 2011]

[ITSA] *Intelligent Transportation Society of America*. Disponible [Internet]: <http://www.itsa.org> [julio de 2011]

[Java] *Java*. Oracle. Disponible [Internet]: <http://www.java.com> [julio de 2011]

[JOSM] *JOSM: Extensible Editor for OpenStreetMap*. Disponible [Internet]: <http://www.josm.openstreetmap.de/> [enero de 2012]

[Kar11] Georgios Karagiannis, Onur Altintas, Eylem Ekici, Geert Heijenk, Boangoat Jarupan, Kenneth Lin, Tomothy Weil. *Vehicular Networking: A Survey and Tutorial on Requirements, Architectures, Challenges, Standards and Solutions.* IEEE Communications Surveys & Tutorials. 2011.

[Kra02] Daniel Krajzewicz, Georg Hertkorn, Peter Wagner and Christian Rösel. *SUMO (Simulation of Urban Mobility), an open-source traffic simulation*. German Aerospace Centre (Berlin, Alemania) y Centre of Applied Informatics (Colonia, Alemania). 2002

[Kra98] Stefan Krauß. *Microscopic Modeling of Traffic Flow: Investigation of collision free vehicles dynamics*. Hauptabteilung Mibilität und Systechnik des DLR Köln. 1998

[Lee00] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steve McCanne, Kannan Varadhan, Ya Xu, Haobo Yu. *Advances in network simulation*. The VINT Project. 2000

[MLP09] Zeinab Movahedi, Rami Langar, Guy Pujolle. *A Comprehensive Overview of Vehicular Ad-hoc Network Evaluation Alternatives*. Universidad de París. 2009.

[MSDN] *Microsoft MSDN Library*. Visual Studio 2005. Disponible [Internet]: <http:// http://msdn.microsoft.com/es-es/library/zebw5zk9%28en-us,VS.80%29.aspx> [julio de 2011]

[Mus06] M. Musolesi and C. Mascolo. *A Community based Mobility Model for Ad-hoc Network Research*. Proc. 2 ACM/SIGMOBILE International Workshop on Multi-hop Ad-hoc Networks: from theory to reality. Florencia, Italia. 2006

[Nar08] José E. Naranjo, Felipe Jiménez, José M. Armingol, Arturo de la Escalera. *Entornos inteligentes basados en redes inalámbricas: aplicaciones al transporte, automóvil inteligente/conectado y seguridad vial.* CITIC, Universidad Politécnica de Madrid. Plataforma Prometeo. 2008.

[NS2] The Network Simulator Project: ns-2. Disponible [Internet]: <http://nsnam.isi.edu/nsnam/index.php/User_Information> [diciembre de 2011]

[OMNeT] *OMNeT++ Discrete Network Simulation Framework*. OMNeT Community. Disponible [Internet]: <http://www.omnetpp.org> [noviembre de 2011]

[OMNManual] *OMNeT++ Discrete Event Simulation System Version 4.0 User Manual.* OMNeT Community. Disponible [Internet]: <http://www.omnetpp.org/doc/omnetpp/manual/usman.html> [abril de 2012]

[OSMap] *Open Street Map Project*. Disponible [Internet]: <http://www.openstreetmap.org/> [enero de 2012]

[PPL] *The Perl Programming Language*. Disponible [Internet]: <http://perl.org> [julio de 2011]

[RML] *MIT Media Lab: Reality Mining*. Disponible [Internet]: <http://reality.media.mit.edu/> [febrero de 2012]

[RVT] *Realistic Vehicular Traces*. Disponible [Internet]: <http://lst.inf.ethz.ch/ad-hoc/car-traces/> [febrero de 2012]

[SBP] *Strawberry Perl for Windows*. Disponible [Internet]: <http://strawberryperl.com> [julio de 2011]

[Sdd08] C. Sommer, I. Dietrich and F. Dressler. *A Simulation Model of DYMO for Ad-hoc Routing in OMNeT++*. ACM/ICST International Workshop OMNeT++. 2008

[SDoc] *SUMO User Documentation*. SUMO Community. Disponible [Internet]: <http://sumo.sourceforge.net/doc/current/docs/SUMO_User_Documentation.html> [octubre de 2011]

[SDown] *SUMO Homepage*. SUMO Community. Disponible [Internet]: <http://sumo.sourceforge.net/ > [octubre de 2011]

[SMARTEST] *Simulation Modelling Applied to Road Transport* – European Scheme – Tests. 1999. Disponible [Internet]: <http://www.its.leeds.ac.uk/projects/smartest/ [marzo de 2012]

[Som08] Christoph Sommer, Zheng Yao, Reinhard German and Falko Dressler. *On the Need for Bidirectional Coupling of Road Traffic Microsimulation and Network.* Computer Networks and Communication Systems, Department of Computer Science, University of Erlangen, Germany. 2008

[Som11] Christoph Sommer, David Eckhoff, Reinhard German and Falko Dressler. *A Computationally Inexpensive Empirical Model of IEEE 802.11p Radio Shadowing in Urban Environments*. Computer Networks and Communication Systems Dept. of Computer Science, University of Erlangen, Germany. 2011

[StreetW] *StreetWise, a blog on Transportation*. Disponible [Internet]: <http:// http://streetwise.kittelson.com/> [octubre de 2011]

[SUMO] *SUMO at a glance*. SUMO Community Wiki. Disponible [Internet]: <http://sourceforge.net/apps/mediawiki/sumo/index.php> [octubre de 2011]

[UDel] *UDel Models for Simulation of Urban Mobile Wireless Networks*. Disponible [Internet]: <http://www.udelmodels.eecis.udel.edu/ > [febrero de 2012]

[USCB] TIGER Data Base. *United States Census Bureau*. Disponible [Internet]: <http://www.census.gov/geo/www/tiger/> [diciembre de 2011]

[Var01] András Varga. *The OMNeT++ Discrete Event Simulation System*. Department of Telecommunications, Budapest University of Technology and Economics, Budapest (Hungary). 2001

[VII] *Vehicle Infrastructure Integration*. Disponible [Internet]: <http://www.its.dot.gov/vii/> [enero de 2012]

[WAVE] Wireless Access for the Vehicular Environment (WAVE). Disponible [Internet]: <http://grouper.ieee.org/groups/802/11/Reports/tgp_update.htm> [enero de 2012]

[WAVE2] IEEE *1609 - Family of Standards for Wireless Access in Vehicular Environments (WAVE)*. U.S. Department of Transportation. January 9, 2006

[Weg08] Alex Wegener, Michal Piorkowski, Maxim Raya, Horst Hellbrück, Stefan Fischer and Jean-Pierre Hubaux. *TraCI: An Interface for Coupling Road Traffic and*

Network Simulators. Institute of Telematics, University of Luebeck (Germany); LCA Lab EPFL Lausanne (Switzerland). 2008

[Zim05] H. M. Zimmermann, I. Gruber. *A Voronoi-based Mobility Model for Urban Environments*. Proc. European Wireless ’05. 2005

ANEXO
Código ejemplo de simulación en VEINS

a. misNodos.nod.xml

```
<nodes>

<node id="oeste" x="300.0" y="+500.0" />
<node id="este" x="+700.0" y="+500.0" />
<node id="norte" x="+500.0" y="+700.0" />
<node id="sur" x="+500.0" y="300.0"/>
<node id="sur2" x="+500.0" y="350.0"/>
<node id="oeste2" x="+350.0" y="500.0"/>
<node id="centro" x="+500.0" y="+500.0" type="priority" />

</nodes>
```

b. misEnlaces.edg.xml

```
<edges>

<edge id="norte_centro" fromnode="norte" tonode="centro" type="tipo1" />
<edge id="este_centro" fromnode="este" tonode="centro" type="tipo2" />
<edge id="centro_oeste2" fromnode="centro" tonode="oeste2" type="tipo2"
/>
<edge id="centro_sur2" fromnode="centro" tonode="sur2" type="tipo1" />
<edge id="sur2_sur" fromnode="sur2" tonode="sur" type="tipo1" />
<edge id="oeste2_oeste" fromnode="oeste2" tonode="oeste" type="tipo1" />

</edges>
```

c. misTipos.typ.xml

```
<types>

<type id="tipo1" priority="1" nolanes="2" speed="10.0" />
<type id="tipo2" priority="2" nolanes="2" speed="5.0" />
</types>
```

d. misConexiones.con.xml

```
<connections>

<connection from="norte_centro" to="centro_oeste2" fromLane="0"
toLane="0" />
<connection from="norte_centro" to="centro_sur2" fromLane="0" toLane="0"
/>
<connection from="este_centro" to="centro_oeste2" fromLane="0"
toLane="0" />
<connection from="este_centro" to="centro_sur2" fromLane="0" toLane="0"
/>
<connection from="centro_sur2" to="sur2_sur" fromLane="0" toLane="0" />
<connection from="centro_oeste2" to="oeste2_oeste" fromLane="0"
toLane="0" />

</connections>
```

e. misRutas.rou.xml

```
<routes>

<vtypeDistribution id="dist_tipo1">
<vtype id="tipo1" accel="0.8" length="5" maxspeed="20"
probability="0.6"/>
<vtype id="tipo2" accel="1.8" length="15" maxspeed="15"
probability="0.1"/>
<vtype id="tipo3" accel="1.2" length="8" maxspeed="25"
probability="0.3"/>
</vtypeDistribution>

<route id="ruta0" edges="norte_centro centro_sur2 sur2_sur"/>
<route id="ruta1" edges="norte_centro centro_oeste2 oeste2_oeste"/>
<route id="ruta2" edges="este_centro centro_oeste2 oeste2_oeste"/>
<route id="ruta3" edges="este_centro centro_sur2 sur2_sur"/>

<flow id="flujo1" route="ruta0" begin="0" e="100" no="100000"
type="dist_tipo1"/>
<flow id="flujo2" route="ruta1" begin="0" e="100" no="100000"
type="dist_tipo1"/>
<flow id="flujo3" route="ruta2" begin="0" e="100" no="100000"
type="dist_tipo1"/>
<flow id="flujo4" route="ruta3" begin="0" e="100" no="100000"
type="dist_tipo1"/>

</routes>
```

f. TraCIDemo.cc

```
#include "applications/traci/TraCIDemo.h"

#include "NotificationBoard.h"
#include "UDPSocket.h"

Define_Module(TraCIDemo);

void TraCIDemo::initialize(int stage) {
    BasicModule::initialize(stage);
    if (stage == 0) {
        debug = par("debug");

        traci = TraCIMobilityAccess().get();
        sentMessage = false;

        NotificationBoard* nb = NotificationBoardAccess().get();
        nb->subscribe(this, NF_HOSTPOSITION_UPDATED);

        setupLowerLayer();
    }
}

void TraCIDemo::setupLowerLayer() {
    cMessage *msg = new cMessage("UDP_C_BIND", UDP_C_BIND);
    UDPControlInfo *ctrl = new UDPControlInfo();
    ctrl->setSrcPort(12345);
    ctrl->setSockId(UDPSocket::generateSocketId());
    msg->setControlInfo(ctrl);
    send(msg, "udp$o");
}

void TraCIDemo::handleMessage(cMessage* msg) {
    if (msg->isSelfMessage()) {
        handleSelfMsg(msg);
    } else {
        handleLowerMsg(msg);
    }
}

void TraCIDemo::handleSelfMsg(cMessage* msg) {
}

void TraCIDemo::handleLowerMsg(cMessage* msg) {
    if (!sentMessage) sendMessage();
    delete msg;
}

void TraCIDemo::receiveChangeNotification(int category, const
cPolymorphic *details) {
    Enter_Method_Silent();

    if (category == NF_HOSTPOSITION_UPDATED) {
        handlePositionUpdate();
    }
}

void TraCIDemo::sendMessage() {
```

```
    sentMessage = true;

    cPacket* newMessage = new cPacket();

    newMessage->setKind(UDP_C_DATA);
    UDPControlInfo *ctrl = new UDPControlInfo();
    ctrl->setSrcPort(12345);
    ctrl->setDestAddr(IPAddress::ALL_HOSTS_MCAST);
    ctrl->setDestPort(12345);
    delete(newMessage->removeControlInfo());
    newMessage->setControlInfo(ctrl);

    sendDelayed(newMessage, 0.010, "udp$o");
}

void TraCIDemo::handlePositionUpdate() {
    if (traci->getPosition().x < 7350) {
        if (!sentMessage) sendMessage();
    }
}
```

PRESUPUESTO

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor: PABLO GONZÁLEZ-RIPOLL CEREZO

2.- Departamento: INGENIERÍA TELEMÁTICA

3.- Descripción del Proyecto:

- Titulo **Estudio Práctico del Simulador de Redes Vehiculares VEINS**
- Duración (meses) **12**

Tasa de costes Indirectos: **20%**

4.- Presupuesto total del Proyecto (valores en Euros):

23.623 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F.	Categoría	Dedicación (hombres mes) [a)]	Coste hombre mes	Coste (Euro)
Pablo González-Ripoll Cerezo	X	Ingeniero	7	2.694,39	18.860,73
		Hombres mes 7		**Total**	18.860,73

[a)] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d)]
PC Sony VAIO	1.000,00	100	12	60	200,00
				Total	200,00

[d)] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} \text{xCxD}$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
-	-	-
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e)]

Descripción	Empresa	Costes imputable
Publicaciones	IEEE y otras instituciones	25 publicaciones x 25,00
	Total	625,00

[e)] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,…

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	18.861
Amortización	200
Subcontratación de tareas	0
Otros costes directos	625
Costes Indirectos	3.937
Total €	23.623