Universidad Zaragoza
1542

Trabajo Fin de Grado

Implementación de un modelo de consumo de vehículos eléctricos en el simulador ns-3

Implementation of an electric vehicle consumption model in the ns-3 simulator

Autor

Samuel Salvatella Pérez

Director

Francisco J. Martínez Domínguez

Escuela Universitaria Politécnica de Teruel

2018

Obra: Implementación de un modelo de consumo de vehículos eléctricos en el simulador ns-3.

Autor: Samuel Salvatella Pérez

Para cualquier duda o sugerencia puede contactar con el autor mediante la siguiente dirección email:

ssalvatellaperez@gmail.com

RESUMEN

En los últimos años, estamos presenciando una revolución tanto en la forma en la que nos movemos, como en las preferencias a la hora de adquirir un vehículo. Los continuos cambios en el precio del petróleo y sus derivados, así como la preocupación creciente debido al aumento de la contaminación, sobre todo en las grandes ciudades, está propiciando el uso de otros combustibles y fuentes de energía mucho más respetuosas con el medio ambiente.

Los vehículos equipados con los tradicionales motores de combustión están dando paso a los nuevos motores, fundamentalmente eléctricos, cuyas ventajas están permitiendo que el número de ventas a nivel mundial esté creciendo continuamente.

En este trabajo fin de grado, nos centramos en los vehículos eléctricos, describimos sus orígenes, presentamos su situación actual, y comentamos brevemente los retos a los que se enfrentan. Dado que, en el futuro más próximo, en el que se espera que el número de vehículos eléctricos crezca todavía mucho más, la energía necesaria para cargar esto vehículos puede representar un problema. Por ello, el uso de la simulación nos va a permitir predecir cómo afectará y qué ventajas tendremos cuando este tipo de vehículos se imponga en el mercado.

En el presente trabajo, presentamos un modelo de consumo para los vehículos eléctricos y lo incluimos en el simulador ns-3, un simulador de redes de código abierto. En concreto, este modelo calcula el consumo eléctrico de los vehículos en función de su movilidad. El resultado final es que el simulador ns-3 permite estimar de forma precisa el consumo de los vehículos eléctricos a partir de las trazas de movilidad, y presenta unos resultados muy similares a otros simuladores ampliamente utilizados y validados por la comunidad científica, como el simulador de movilidad SUMO. Además, el código realizado durante nuestro trabajo, vamos a incorporarlo a la distribución oficial del simulador ns-3, lo que permitirá a los investigadores de todo el mundo reutilizar las trazas de movilidad generadas con anterioridad, así como realizar nuevas propuestas relacionadas con los vehículos eléctricos y las comunicaciones vehiculares.

Palabras clave: vehículos eléctricos, simulación, modelo de consumo, ns-3

ÍNDICE

1 INTRODUCCIÓN

La crisis ecológica mundial causada por el calentamiento global es un hecho actualmente innegable. La comunidad científica en su totalidad concluye que el ser humano es el principal responsable del calentamiento global [1]. Del total de las emisiones de CO_2 causantes del efecto invernadero, un 20% son generadas por el transporte (datos de 2014 [2]). Afrontar el desafío de frenar el empeoramiento de la situación conlleva obligatoriamente la transformación de la movilidad tanto en ciudades como en grandes distancias. Buscar la reducción de emisiones, junto a las numerosas crisis del petróleo, ha hecho resurgir el vehículo eléctrico.

Aunque el vehículo eléctrico fue uno de los primeros en aparecer sobre las carreteras, durante el siglo XX desapareció completamente. En las últimas décadas ha vuelto a aparecer en escena como una alternativa viable a la movilidad convencional basada en combustibles fósiles. Aun así, es una tecnología que presenta varios problemas que deben de ser solucionados para poder consolidarse. Entre los problemas a destacar están la autonomía, el consumo eléctrico a gran escala o la recarga de los vehículos.

Encontrar las soluciones a estas dificultades es un trabajo, por su naturaleza, multidisciplinar que abarca un gran conjunto de ramas y técnicas. Una de estas técnicas es la recolección de datos para su posterior estudio. En el siguiente trabajo, se ha optado por aportar herramientas al análisis y simulación de datos. La simulación permite la obtención de una gran cantidad de datos realistas de manera rápida y asequible. Esto nos da la opción de analizar posibles escenarios en los que esté implicada la movilidad de vehículos eléctricos para la investigación de distintos ámbitos, como por ejemplo la elección de puntos de recarga o las redes VANET (del inglés *Vehicular Ad-Hoc Network*).

2 OBJETIVOS

Para realizar el siguiente trabajo se han propuesto los siguientes objetivos. Primero, una serie de objetivos generales que encauzan en cierta forma la filosofía seguida. Posteriormente, se han definido a partir de los generales, unos objetivos concretos que especifican el resultado de haber completado el trabajo.

2.1 GENERALES

I. Estudiar y conocer cuáles es la situación real del vehículo eléctrico en estos momentos y a qué desafíos se enfrenta su desarrollo.

II. Aportar herramientas para la mejora de las tecnologías implicadas en el vehículo eléctrico.

III. Participar en un proyecto *open-source* relacionado con la problemática.

2.2 CONCRETOS

Cada uno de estos objetivos generales más abstractos se traducen en objetivos concretos.

1. Redactar un estado del arte que especifique cuál es la situación del vehículo eléctrico tanto a nivel global como a nivel del estado español. Encontrar también cuáles son los desafíos tecnológicos a los que se enfrenta su implantación.

2. Diseñar e implementar un módulo para el consumo de vehículos dentro del simulador ns-3 que incorpore un modelo para calcular concretamente el consumo de vehículos eléctricos.

3. Comprobar los resultados del modelo implementado comparándolos con otros simuladores como SUMO (del inglés *Simulation of Urban MObility*) y medir la precisión de lo implementado.

4. Conseguir que la implementación realizada sea aceptada en el código base del proyecto ns-3, aunque éste es un objetivo más a largo plazo que depende de terceros.

3 ESTADO DEL ARTE DEL VEHÍCULO ELÉCTRICO

Con el fin de resaltar y entender la importancia de la investigación en el campo escogido para este trabajo, hay que analizar cuál es la situación de los vehículos eléctricos actualmente, así como toda la tecnología relacionada con éstos. Realizar este estado del arte nos permite también conocer cuáles son los desafíos y problemáticas que afronta el desarrollo de tecnologías para una movilidad sostenible. De esta manera, se dota al trabajo tanto de una base teórica como de una justificación, dada la necesidad de aportar soluciones en varias cuestiones que se comentarán posteriormente.

3.1 HISTORIA

El origen de los primeros vehículos eléctricos se remonta al descubrimiento del electromagnetismo en 1820. Su invención se atribuye a varias personas. En 1827 el eslovaco-húngaro Ányos Jedlik construyó el primer motor eléctrico primitivo que usaría un año después para mover un pequeño coche de juguete. Posteriormente, en 1835, el profesor Sibrandus

Stratingh diseñó un coche eléctrico a pequeña escala propulsado por pilas eléctricas no recargables [3]. El avance en la creación de motores eléctricos trajo consigo la aparición de la primera locomotora impulsada por energía eléctrica en 1837 creada por el químico Robert Davidson. Más tarde, fabricaría una versión a gran escala probada en 1842 en la línea de Edimburgo-Glasgow. Las líneas de tranvías y trolebuses impulsados por motores eléctricos llegarían a las principales ciudades europeas a finales del siglo XIX, aunque en lugar de baterías, obtendrían la energía de catenarias y raíles conductores [4].

Las limitaciones de los vehículos eléctricos impulsados por baterías hacían imposible su uso con fines prácticos o de explotación. Las primeras fuentes de energía eran células electroquímicas basadas en zinc-platino inventadas por el químico gales William R. Grove en 1830 y más tarde en células despolarizadas de doble electrolito desarrolladas por el inglés John Daniell en 1836. Estas baterías debían de ser cambiadas cada vez que se agotaban por lo que resultaban extremadamente caras, de manera que los primeros vehículos eléctricos no tuvieron el éxito esperado. A pesar de ello, estos primeros modelos presentaban más ventajas frente a los vehículos de combustión que no mejorarían hasta 1863 con la invención del motor de combustión interna de Etienne Lenoir [3].

No fue hasta 1881 que llegaría el primer vehículo eléctrico que funcionase con baterías recargables inventado por el francés Gustave Trouvé (Ver Figura 1). En la Exposición Internacional de Electricidad mostraría su automóvil de tres ruedas con las baterías diseñadas por el belga Gaston Plante en 1859 que podían ser cargadas después de su uso [5]. De esta manera, comenzó la aparición de modelos con mejores prestaciones impulsando una industria de producción de vehículos eléctricos. El primer vehículo comercial sería un taxi eléctrico en la ciudad de Nueva York en 1897 [6]. Los vehículos eléctricos conformarían en ese momento el 28% del total de vehículos en carretera [7].

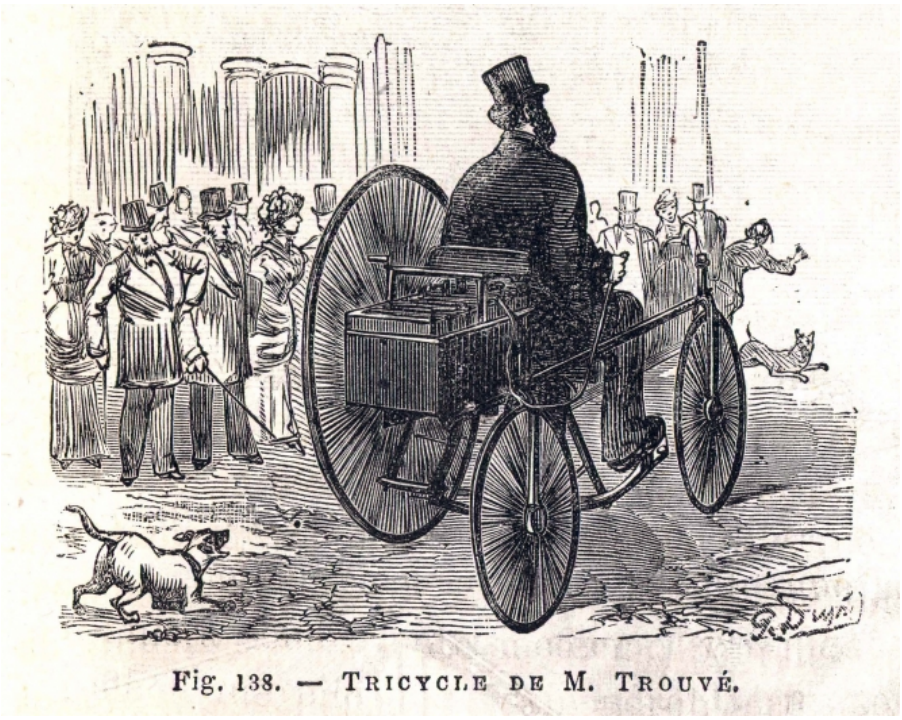


Figura 1 – Primer vehículo eléctrico con baterías recargables de Gustave Trouvé [Fuente: : https://www.connexionfrance.com/Mag/French-Facts/Introducing-the-French-inventor-of-the-electric-car]

Sin embargo, a partir de 1908 los vehículos impulsados por gasolina ganaron una gran popularidad gracias al modelo Ford T. El abaratamiento del petróleo conllevó unos costes de uso mucho menores para los vehículos de gasolina frente a los costes de uso de los vehículos eléctricos. Por otra parte, el alcance y autonomía tan limitados de los vehículos eléctricos contribuyó también a su descenso de popularidad, llegando a su práctica desaparición de las carreteras en 1935 [6]. No sería hasta décadas después, en los 90, con el encarecimiento del petróleo y la necesidad de atajar el cambio climático, que resurgiría el interés por los vehículos eléctricos y su desarrollo.

3.2 ESTADO ACTUAL

En los últimos 15 años, el interés en la electrificación del transporte ha aumentado considerablemente, en busca de la reducción de las emisiones contaminantes que contribuyen al cambio climático y el empeoramiento de la calidad del aire. Este renovado interés inicia, según la Agencia Internacional de la Energía, una "tercera edad" de los vehículos eléctricos con su introducción al mercado masivo [7].

A continuación, se presenta por un lado la situación global de los vehículos eléctricos y por otro, de manera más específica, la situación en España. Nótese que se hace la distinción en los datos entre vehículo eléctrico de batería o BEV (del inglés *battery-electric vehicle*) y vehículo híbrido eléctrico enchufable o PHEV (del inglés *plug-in hybrid electric vehicle*). El

primero solo es impulsado por motores eléctricos alimentados por baterías, mientras que el segundo posee aparte de un motor eléctrico, otro de combustión interna y unas baterías que permiten ser recargadas enchufándolas a la red.

3.2.1 Situación global

En 2016 se vendieron mundialmente 753.000 vehículos eléctricos de los cuales el 60% eran BEV. Esto ha supuesto un aumento de las ventas del 40% estableciendo un récord histórico de dos millones de coches eléctricos en circulación en todo el mundo [8]. A pesar de ello, el número de vehículos eléctricos supone solamente el 0,2% de todos los coches en circulación.

China es el país con mayor número de vehículos eléctricos (BEV y PHEV) en circulación con 648.770 unidades. Le siguen Estados-Unidos con 563.710 vehículos eléctricos y Japón con 151.250. Por último, Noruega (133.260 unidades) y Países Bajos (112.010 unidades) seguirían el ranking de países con más coches eléctricos en circulación (año 2016). (Ver Tabla 1)

Posición	*País*	*N º de vehículos eléctricos*
1º	China	648,77
2º	Estados-Unidos	563,71
3º	Japón	151,25
4º	Noruega	133,26
5º	Países-Bajos	112,01
6º	Otros	87,48
7º	Reino-Unido	86,42
8º	Francia	84,00
9º	Alemania	72,73
10º	Suecia	29,33
11º	Canadá	29,27
12º	Corea del Sur	11,21
13º	India	4,80

Tabla 1 – Ranking mundial de países con mayor número de vehículos eléctricos en circulación (en miles) [9].

Si se habla de la cuota de mercado que suponen los vehículos eléctricos en función del país, Noruega lidera el ranking con un 28,76%. Datos sorprendentemente buenos si los comparamos con el resto de los países. Le seguirían, en segundo y tercer puesto, los Países Bajos con un 6,39% y Suecia con el 3,41%. Francia con un 1,46% de cuota de mercado y Reino Unido con el 1,41% cerrarían la lista de los cinco países con mayor cuota de mercado de vehículos eléctricos. (Ver Tabla 2)

Posición	*País*	*Cuota de mercado*
1º	Noruega	28,76%
2º	Países-Bajos	6,39%
3º	Suecia	3,41%
4º	Francia	1,46%
5º	Reino-Unido	1,41%
6º	China	1,37%
7º	Estados-Unidos	0,91%
8º	Alemania	0,73%
9º	Japón y Canadá	0,59%
10º	Otros	0,52%
11º	Corea del Sur	0,34%
12º	India	0,02%

Tabla 2 – Ranking mundial de países en función de la cuota de mercado del vehículo eléctrico (%) [9].

Respecto a los modelos y marcas de coches eléctricos, los que registran el mayor número de ventas son: el Chevrolet Volt con 109.472 ventas en primer lugar y después el modelo Nissan Leaf con 101.679 unidades. En tercer lugar, el modelo Tesla S con 85.762 ventas y le siguen el Toyota Prius Plug-in (42.345 unidades) y el Ford Fusion Energi (42.228 ventas) [10].

El crecimiento del sector del vehículo eléctrico no sería posible sin la voluntad y acción de varios países. En 2009, se creó el foro por la iniciativa de vehículos eléctricos o EVI (del inglés *Electric Vehicles Iniciative*) formado por varios gobiernos mundiales con el objetivo de acelerar el desarrollo de la movilidad eléctrica a nivel mundial [11]. En este momento, los países miembros de esta iniciativa son: Canadá, China, Francia, Alemania, Japón, Países-Bajos, Noruega, Suecia, Reino-Unido y Estados-Unidos. Esta iniciativa tiene dos objetivos principales. Por un lado, recoger y agrupar datos sobre la situación mundial del vehículo eléctrico (ventas, políticas, desarrollo de infraestructuras) para su análisis. Y por otro, favorecer y asistir la aplicación de políticas que incentiven la expansión de los coches eléctricos globalmente [12]. La EVI estableció el objetivo global de 20 millones de vehículos eléctricos en las carreteras para el año 2020. Este objetivo es demasiado ambicioso si se tiene en cuenta que los fabricantes de coches prevén 13 millones de coches eléctricos en 2020 y recientemente en 2017 se ha llegado a los 2 millones [13].

A modo de balance, el progreso en la implantación de vehículos eléctricos es positivo sobre todo dentro de los participantes en la EVI que representan el 95% de las ventas realizadas en el mundo. Un 40% de crecimiento anual es un buen dato, aunque el crecimiento no puede desacelerarse a partir de ahora si se quiere alcanzar los objetivos para el 2025. Estamos por lo tanto en un momento crucial para consolidar un futuro con

una movilidad sostenible. Para ello, los gobiernos y países deben de comprometerse y seguir promoviendo la implantación del vehículo eléctrico en las carreteras para frenar lo máximo posible el cambio climático.

3.2.2 Situación en España

En cuanto al estado de la implantación del vehículo eléctrico en España se puede dividir en tres aspectos fundamentales: el número de ventas y cuota en el mercado, la infraestructura de recarga, y las medidas políticas para fomentar el crecimiento del vehículo eléctrico.

Según el informe anual de la ANFAC (Asociación Española de Fabricantes de Automóviles y Camiones), en 2016 se matricularon en España 3.516 vehículos eléctricos, de los cuales, 2.005 eran BEV y 1.511 PHEV. Respecto al año 2015 supuso un aumento del 49,40% en BEV y un 92,48% para PHEV [14]. A pesar del crecimiento anual, el vehículo eléctrico en España supone solamente un 0,3% de la cuota de mercado[1] [15]. Esto nos sitúa en la cola de Europa en cuanto a la importancia del coche eléctrico en nuestro mercado nacional junto a Italia, Estonia o Lituania. La ANFAC establece un objetivo para 2020 de una cuota de mercado del 10% [16]. Este objetivo es claramente irrealizable en solo tres años teniendo en cuenta la cuota actual. Irónicamente, España es uno de los mayores productores de vehículos eléctricos en Europa con 9.257 unidades producidas (BEV y PHEV) en 2016. Los modelos eléctricos que se producen actualmente en España son: Renault Twizy (45 y 85), Citroen Berlingo Eléctrica, Peugeot Partner Eléctrica, Nissan e-Nv 200, Nissan EVALIA, y Ford Mondeo Hybrid.

Actualmente en todo el territorio de España existen 2.806 puntos de recarga de vehículos eléctricos. De estos puntos de recarga, solamente 729 son de uso público. El resto de los puntos se reparten entre parkings de centros comerciales, restaurantes, hoteles, etc. También es significativo que solamente 102 puntos se encuentran en estaciones de servicio [17]. A todo esto, hay que añadir que existen varios tipos de enchufes y no todos los vehículos son compatibles con todos los puntos de recarga. La mayoría de los cargadores se concentran en las grandes ciudades. Aunque pueda parecer lógico que se concentren los puntos de recarga en las ciudades, resulta ineficiente y tiene poco sentido si pensamos en las necesidades del vehículo eléctrico. Al tener poca autonomía (entre 200 y 400 km) los puntos de recarga resultan especialmente útiles en los caminos entre las grandes ciudades y no dentro de éstas. Esto significa que los viajes largos para un vehículo eléctrico

[1] Existe una discrepancia sobre el valor real de la cuota de mercado del vehículo eléctrico en España. Mientras que la Asociación española de Fabricantes de Automóviles y Camiones estima este valor en un 2,6 % en su informe anual [14], la Asociación Europea de Fabricantes de Automóviles lo estima en un 0,3 % [15]. Se prefiere utilizar este último valor puesto que tiene más sentido, teniendo en cuenta la posición de España en el ranking mundial.

son difíciles. Según Xavier Cañadell, consejero delegado de Electromaps, "*Es imposible ir de Madrid a Barcelona"* [18].

En el año 2014, el gobierno de España estableció el Plan de Impulso a la Movilidad con Vehículos de Energías Alternativas conocido como MOVELE. Se trata de un plan de ayudas económicas que ayuden en la adquisición de vehículos eléctricos y la implantación de más puntos de recarga [19]. Este sería el primer plan económico en España para apoyar la implantación del coche eléctrico. Desde entonces, el plan se ha renovado cada año y en 2016 se aprobó el plan MOVEA 2016, con el mismo objetivo, con un presupuesto de 16,6 millones de euros [20]. Posteriormente para el año 2017 se aprobó un nuevo plan MOVEA, pero con una reducción de presupuesto dejándolo en 14,26 millones de euros [21]. Recientemente, el secretario de Estado de Energía, Daniel Navia, anunció la creación del Plan ProMovea que estará dotado de 50 millones de euros, de los cuales, 20 millones estarán destinados al incentivo de la compra de vehículos eléctricos, y los 30 restantes para la mejora de la infraestructura de carga e incentivar proyectos industriales relacionados [22]. El aumento de presupuesto para este tipo de planes es un dato positivo, aunque puede seguir siendo muy ajustado si lo comparamos con los países vecinos como Francia con partidas de 388 millones de euros para el 2018 [23] o con Alemania que ha presupuestado en 1.200 millones de euros sus ayudas para vehículos eléctricos durante cuatro años [24]. La propia Asociación Española de Fabricantes de Automóviles y Camiones considera imprescindible ampliar los fondos para potenciar el vehículo alternativo [25].

Actualmente, la situación en España del vehículo eléctrico continúa siendo bastante precaria. La cuota de mercado actual convierte a los coches eléctricos en algo anecdótico en las carreteras. La implantación de la movilidad eléctrica se ve perjudicada por una infraestructura de recarga escasa e ineficiente. Los compromisos e iniciativas políticas siguen siendo poco ambiciosos y se quedan muy por detrás de otros países vecinos europeos. En definitiva, falta una apuesta seria por la transformación hacia una movilidad eléctrica que reduzca los efectos del cambio climático y mejore la calidad de vida en las ciudades. Esta apuesta deberá, no solo apoyar mediante ayudas económicas el uso de vehículos eléctricos, sino también apoyar la investigación tanto en los propios vehículos eléctricos como en la infraestructura de carga.

3.3 DESAFÍOS Y PERSPECTIVAS DE FUTURO

El desarrollo e implantación de los vehículos eléctricos a nivel global pasan por la puesta de atención en dos frentes distintos. El primero: la mejora de las tecnologías implicadas, especialmente, las baterías, la red eléctrica y la infraestructura de carga [6]. El segundo: el aspecto financiero y político. Numerosos países interesados en cumplir los objetivos contra el cambio

climático llevan a cabo políticas de apoyo financiero en la compra de vehículos eléctricos mediante ventajas fiscales o con políticas de penalización de vehículos contaminantes [13].

Para que el vehículo eléctrico pueda convertirse en una alternativa real al transporte tradicional se debe llevar acabo un progreso y desarrollo tecnológico continuado. Los mayores obstáculos para la implantación del coche eléctrico son sus propias limitaciones tecnológicas como la baja autonomía, la falta de infraestructura de carga o los tiempos de recarga altos. Existen tres cuestiones esenciales en las que en estos momentos está el foco de atención de las investigaciones: la red eléctrica, las baterías de los vehículos, y la carga eléctrica.

Red eléctrica

La transformación de la movilidad tradicional a una eléctrica pasa inevitablemente por un aumento muy considerable de la demanda eléctrica. Un vehículo eléctrico que realiza 40 km diarios necesita entre 6 y 8 kWh de energía [26]. Para hacerse una idea de lo que supone, el consumo medio de una vivienda en España son 15,67 kWh al día [27]. Teniendo en cuenta que de media la producción de energía en el mundo, el 57% se realiza con energías fósiles [28] (datos de 2016), responsables de la emisión del 42% del dióxido de carbono [29], el progreso del vehículo eléctrico deberá siempre de ir de la mano del avance de las energías renovables no contaminantes.

Otro problema derivado del aumento del número de vehículos eléctricos es el pico de demanda eléctrica. Los picos de demanda consisten en horas del día en el que se produce una demanda muy alta de electricidad. Si imaginamos que los usuarios ponen a cargar su vehículo al volver a casa después de su jornada laboral, la probabilidad de sobrecarga de los transformadores locales sería muy alta [26]. La solución pasa por incentivar la carga de los vehículos en horas de poca demanda, con precios más económicos, junto a sistemas inteligentes que controlen cuando se carga el vehículo.

La introducción de los vehículos eléctricos en la red eléctrica puede aportar también nuevas oportunidades. Hasta ahora solo se ha considerado la recarga de las baterías en la red, pero también se podría usar de manera inversa, es decir, que los vehículos eléctricos pudieran descargarse aportando energía a la red. Esta capacidad se conoce como *vehículo a red* o *Vehicle-to-Grid* (V2G) [6]. Esto permite estabilizar la red cuando ocurren picos de demanda, algo especialmente importante teniendo en cuenta que las energías renovables (solar, eólica, etc.) no producen energía de forma constante. Esta oportunidad puede aprovecharse dentro del desarrollo de una red eléctrica inteligente. La red inteligente (*Smart grid*) consiste en una red moderna que hace uso de las tecnologías de la información y comunicación para mejorar la eficiencia, sostenibilidad y confiabilidad [30]. Los consumidores y productores

de energía de la red se comunican de manera bidireccional obteniendo datos en tiempo real con los que un agente autónomo e inteligente optimiza las operaciones en la red.

Batería

La batería de un vehículo eléctrico es la pieza más importante. Es en ella donde se encuentran las mayores dificultades a la hora del diseño de un coche eléctrico. Hoy en día, la autonomía del vehículo queda muy limitada debido a las baterías actuales que presentan una baja densidad energética (capacidad de energía por kilogramo de peso). Además, se trata del componente más caro y es responsable del alto precio de los vehículos eléctricos. La capacidad total eléctrica no es el único problema. La durabilidad y la seguridad de la batería son otros de los desafíos presentes en su desarrollo.

A pesar de todo, se han realizado enormes avances en las dos últimas décadas mejorando las prestaciones de las baterías eléctricas. Las primeras baterías utilizadas para la tracción de vehículos fueron las compuestas por ácido y plomo. Estas baterías son económicas, pero presentan multitud de desventajas como una muy baja densidad energética, un gran peso, y son extremadamente contaminantes [31].

Enseguida estas baterías fueron sustituidas por unas basadas en níquel e hidruro metálico. Comparadas con las de ácido y plomo, presentan una mayor densidad energética y una buena madurez tecnológica. Aun así, estas baterías presentan desventajas como una eficiencia muy baja en las descargas y cargas, propensión a descargarse solas, y malos resultados en condiciones de frío. Este tipo de baterías se usaron especialmente en la última década en coches eléctricos, pero fueron sustituidas por baterías de tipo Zebra y las más recientes baterías de iones de litio.

Las baterías Zebra, o baterías de sal fundida, utilizan como electrolito tetracloroaluminato de sodio ($NaAlCl_4$) o sodio-níquel-cloro (NaNiCl) y como electrodo negativo sodio fundido. Estas baterías presentan una densidad energética muy alta y una potencia muy alta lo que las hace útil para el uso en vehículos eléctricos. Sin embargo, la batería Zebra necesita una temperatura para operar entre 245 ºC y 350 ºC lo que, junto a la inestabilidad del sodio fundido, provoca muchos problemas de seguridad [32]. El uso de este tipo de baterías en vehículos eléctricos cada vez es más raro.

Las baterías de iones de litio supusieron una revolución para los vehículos eléctricos. Estas baterías ofrecen una gran densidad energética, una alta potencia, no son contaminantes, pueden ser cargadas muy rápidamente y tienen un precio económico. Esto le convierte en el tipo de batería perfecto para los vehículos eléctricos. Actualmente, la gran mayoría de modelos utilizan este tipo de baterías. Aun así, estas baterías tienen alguna

desventaja, al no ser una tecnología completamente madura, como los riesgos de incendio y explosión en caso de ser defectuosa [33].

Recarga del vehículo

La recarga de un vehículo eléctrico se debe realizar mediante un cargador. El cargador es el responsable de transformar la corriente alterna (AC) de la red eléctrica a una corriente continua (DC) que es la que utilizan las baterías. Los cargadores pueden situarse dentro del vehículo con lo que solo tienen que conectarse a la red, pero tienen peores prestaciones como una carga mucho más lenta que un cargador externo preparado para ofrecer un servicio de carga rápida. Existen una serie de estándares internacionales establecidos por distintos organismos como la Sociedad de Ingenieros de Automoción (SAE), la asociación CHAdeMO (acrónimo de *CHArge de MOve*) y la Comisión Electrotécnica Internacional (IEC) [34].

El estándar de SAE EV es un estándar que divide la recarga en tres niveles. El primer nivel está diseñado para una carga lenta desde la red con corriente alterna pensado para cargas nocturnas estimando 17 horas para cargar desde el 20 % de carga a completamente cargado. El segundo nivel es para cargas rápidas capaces de cargar al 80% un vehículo en veinte minutos. Por último, el tercer nivel, aún en desarrollo, propone cargas de gran potencia de hasta 240 kW que cargarían baterías en menos de 10 minutos [35].

El estándar IEC EV categoriza las cargas en tipos de conexión y modos de carga. Establece tres tipos de conexión en función del conector utilizado para realizar la carga. Para los modos de carga, se especifican cuatro modos. El primer modo para cargas lentas pensadas para la red de viviendas. El segundo modo también para cargas lentas en viviendas, pero haciendo uso de dispositivo eléctrico de seguridad. El tercer modo soportaría cargas rápidas y lentas con un dispositivo fijo de control y seguridad. El último modo está diseñado para cargas muy rápidas haciendo uso de un cargador externo. Los tres primeros modos estarían pensados para cargas entre 3 y 10 horas conectados a la red principal de cualquier hogar, mientras que el último modo podría cargar un vehículo en 10 minutos con cargadores externos muy costosos [36].

El estándar CHAdeMO fue creado para estandarizar las cargas rápidas de los vehículos eléctricos. Su nombre es un juego de palabras japonés de la frase "¿Tomamos un té?", en referencia al tiempo que se tardaría en recargar las baterías. Su objetivo es el de acabar con el problema de los viajes largos del vehículo eléctrico permitiendo realizar cargas hasta del 80 % en menos de 30 minutos. Este estándar intenta también mejorar el equilibrio entre el coste de implementarlo y los beneficios. Su éxito ha sido notable y el organismo de la Comisión Electrónica Internacional lo ha adoptado también dentro de sus estándares [37].

La cuestión de la recarga de vehículos eléctricos no es solo una cuestión puramente tecnológica, sino también es ofrecer una infraestructura eficiente que permita cargar el número cada vez más creciente de vehículos eléctricos. Para ello, es importante seleccionar los puntos geográficos más interesantes donde posicionar cargadores, puesto que las necesidades de recarga no son iguales en todos los puntos y pueden variar en función del número de vehículos eléctricos o incluso la orografía. Realizar esta tarea no es posible sin el análisis de datos y la generación de modelos que nos permitan predecir las necesidades de la movilidad eléctrica de manera que se pueda ser capaz de elegir los mejores puntos de recarga futuros eficientemente.

3.4 CONCLUSIONES

El futuro del vehículo eléctrico es prometedor. El número de estos vehículos en el mundo aumenta año por año previendo que en 2020 haya más de 10 millones de éstos en el mundo y realmente estamos entrando en una nueva era para la movilidad eléctrica. El éxito de la implantación del vehículo eléctrico está muy ligado con el progreso tecnológico de éste. La mejora de las baterías, consagrar la red eléctrica como una red inteligente, o mejorar la infraestructura de recarga de los vehículos, son los desafíos más importantes en estos momentos.

La creación de modelos para el análisis de datos puede impulsar las tecnologías del vehículo eléctrico eficientemente. Dentro del análisis de datos hay que señalar la importancia de la herramienta de la simulación. La capacidad de simular escenarios en los que se incluyan vehículos eléctricos supone una reducción de costes al permitir evaluar sus usos sin gasto alguno. Hay que destacar también la rapidez de los datos obtenidos mediante simulación, sin tener que pasar por mediciones en campo, junto con la oportunidad de realizar análisis sobre éstos de manera rápida. El uso de simulaciones nos permite aumentar la viabilidad de la implantación del vehículo eléctrico. En contra parte, la simulación presenta menor precisión y realismo frente a datos reales de campo, aunque esta dependerá siempre de la implementación realizada.

4 MODELO DE CONSUMO ELÉCTRICO

Para lograr implementar un modelo que simule el consumo de vehículos eléctricos en función de su desplazamiento, se va a seguir un enfoque determinista frente a uno estocástico. Es decir, se concibe el consumo del vehículo como parte de un sistema determinista en el que el azar no está involucrado. Naturalmente, se reconoce que pueden existir variables externas que influyan en el consumo exacto del vehículo, pero se asume que una simulación es una aproximación a la realidad y el margen de diferencia para un modelo físico de este tipo es bastante pequeño.

El modelo implementado se ha basado en el especificado por Tamás Kurczveil, Pablo Álvarez López y Eckehard Schnieder que utilizaron en el simulador de tráfico SUMO [38].

Lo primero para calcular el consumo de un vehículo es obtener su energía. Esta la podemos obtener sumando su energía cinética, su energía potencial y su energía rotacional para después sustraer la energía perdida debido a la resistencia de diversos componentes del vehículo. Por lo tanto, la energía del vehículo en un instante k podemos calcularla con la siguiente ecuación:

$$E_{veh}[k] = E_{kin}[k] + E_{pot}[k] + E_{rot,int}[k] = \frac{m}{2} \cdot v^2[k] + m \cdot g \cdot h[k] + \frac{J_{int}}{2} \cdot v^2[k]$$

Una vez calculada la energía del vehículo podemos obtener la energía consumida (o ganada) entre un instante k y $k+1$ restando la energía perdida:

$$\Delta E_{gain}[k] = E_{veh}[k+1] - E_{veh}[k] - \Delta E_{loss}[k]$$

Para calcular la energía perdida total debemos de sumar la energía perdida por estos cuatro componentes: el rozamiento del aire, el rozamiento con el suelo, el rozamiento de curva y la energía constante consumida por el vehículo:

$$\Delta E_{loss}[k] = \Delta E_{air}[k] + \Delta E_{roll}[k] + \Delta E_{curve}[k] + \Delta E_{const}[k]$$

$$\Delta E_{air}[k] = \frac{1}{2} \cdot \rho_{air} \cdot A_{veh} \cdot c_w \cdot v^2[k] \cdot |\Delta s[k]|$$

$$\Delta E_{roll}[k] = c_{roll} \cdot m \cdot g \cdot |\Delta s[k]|$$

$$\Delta E_{curve}[k] = c_{rad} \cdot \frac{m \cdot v^2[k]}{r[k]} \cdot |\Delta s[k]|$$

$$\Delta E_{const}[k] = P_{const} \cdot \Delta t$$

En función de si la energía entre k y $k+1$ es positiva o negativa habrá que multiplicarla por un factor de recuperación o de propulsión, que vendrán determinados por cómo aprovecha el vehículo en cuestión la energía para descargarla o cargarla de la batería. La ecuación para saber el nivel de la batería del vehículo en un instante $k+1$ queda entonces de la siguiente forma:

$$E_{bat}[k+1] = E_{bat}[k] + \Delta E_{gain}[k] \cdot \eta_{prop}$$

$$E_{bat}[k+1] = E_{bat}[k] + \Delta E_{gain}[k] \cdot \eta_{recup}$$

Siendo:

η_{recup} → factor de eficiencia para recuperación	ρ_{air}→ densidad del aire
η_{prop} → factor de eficiencia para propulsión	A_{veh}→ superficie frontal del vehículo
m → masa del vehículo	c_w→ resistencia del aire
$v[k]$→ velocidad del vehículo en instante k	$s[k]$→ distancia cubierta
g→ aceleración gravitacional	c_{roll}→ resistencia a la rodadura
J_{int}→ momento de inercia interno	c_{rad}→ coeficiente de resistencia de la curva P_{const}→ la potencia (promedio) del consumo constante

Al implementar el modelo de consumo eléctrico con estas ecuaciones se han hecho las siguientes **asunciones**:

- La velocidad es constante entre el instante k y $k + 1$.
- La densidad del aire es siempre la misma.
- El consumo eléctrico debido al propio funcionamiento del vehículo (aire acondicionado, luces, etc.) es constante.

Este modelo resulta coherente con la física clásica implicada en el movimiento de un vehículo y es fiable al ser implementado por simuladores de movilidad urbana muy importantes como SUMO. Por ello, ha sido escogido para realizar la implementación dentro del simulador ns-3.

5 SIMULADOR NS-3

El simulador ns-3 es un simulador de redes de comunicación basado en eventos discretos. Es decir, se trata de un sistema de simulación que otorga control en la variable tiempo permitiendo avanzar a éste a intervalos variables. Esto significa que podemos representar modelos compuestos por una secuencia de eventos en el tiempo. En lo que respecta a la simulación de comunicaciones, ns-3 se utiliza para establecer topologías definidas por nodos interrelacionados a los que se les añaden modelos de comunicación (UDP, IPv4, etc.) y modelos de movilidad para su posterior ejecución y análisis de los resultados.

Su origen se remonta a 1989, cuando se empezó a desarrollar la primera versión como una variante del simulador de redes REAL creado por Srinivasan Keshav [39]. Esta primera versión llamada ns-1 ganó el apoyo en 1995 de la Agencia de Proyectos de Investigación Avanzados de Defensa o DARPA (del inglés *Defense Advanced Research Projects Agency*) y otras

agencias de investigación estadounidenses, lo que empujó su desarrolló gracias a sus investigadores. Más tarde, se inició el desarrollo de una nueva versión ns-2**,** como revisión de la anterior, incorporando protocolos de comunicación más modernos como los *wireless*. El desarrollo y mantenimiento del ns-1 se detuvo en el 2001 y el del ns-2 en 2010. La nueva y actual versión del ns-3 surgió en 2005 con la intención de modernizar el simulador empezando desde cero toda la programación. Su desarrollo fue impulsado por el investigador Tom Henderson y fue llevado a cabo principalmente por investigadores de la Universidad de Washington, el Instituto Tecnológico de Georgia y el grupo de investigación Planète en INRIA (del francés *Institut National de Recherche en Informatique et en Automatique*) [40].

Actualmente, ns-3 está en su versión 3.28 y cuenta con más de 38 módulos para distintos tipos de simulación que consagran miles de líneas de código en lenguaje C++. Está en constante desarrollo y cuenta con una comunidad de desarrolladores activa, compuesta principalmente por investigadores de varias universidades. En torno al mantenimiento del simulador se estableció un consorcio compuesto por las distintas organizaciones que lo mantienen y desarrollan. El acuerdo para establecer el consorcio del ns-3 se realizó entre la Universidad de Washington y el INRIA y especifica el funcionamiento de la organización [41]. (Ver Figura 2)

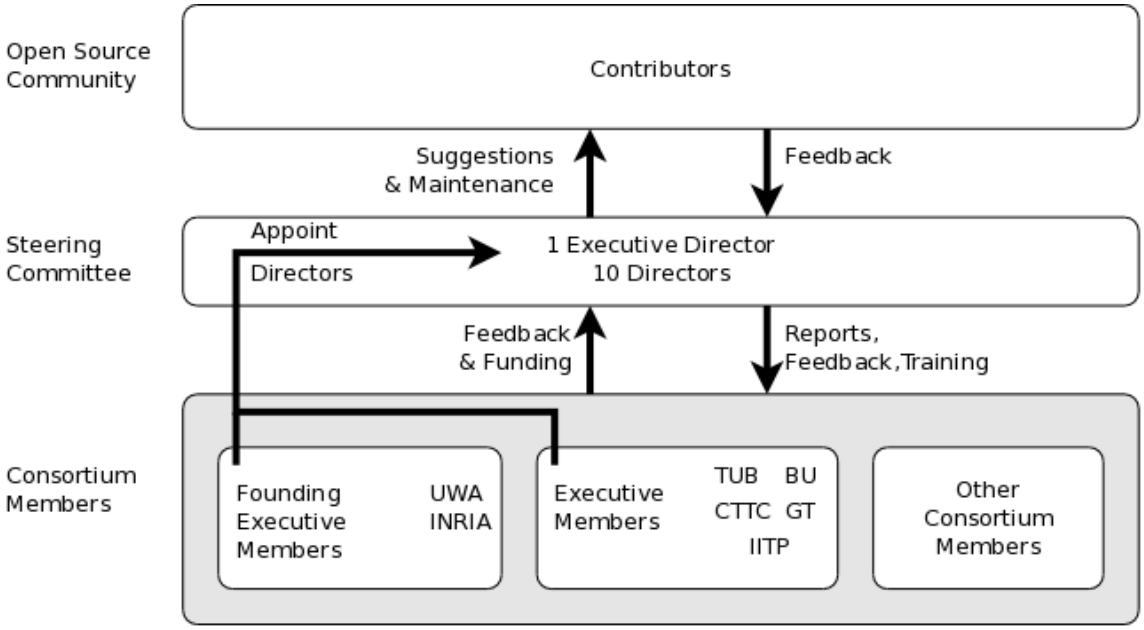


Figura 2 - Organigrama del ns-3 Consortium

5.1 MOTIVACIÓN DE SU USO

Hasta ahora se ha explicado el origen y en qué consiste el simulador ns-3. A continuación, se explicará por qué ha sido escogido para implementar un modelo de consumo de vehículos eléctricos y no se ha creado desde cero como

un programa *standalone* o con otro simulador. La razón por la que se ha escogido este simulador es que no se pretende simular simplemente el consumo de los vehículos (para eso ya existen otros simuladores como SUMO [2]), sino ayudar en la investigación en campos de estudio que comprenden diversas temáticas. Un ejemplo de esto serían las redes VANET (del inglés *Vehicular Ad-Hoc Network*) que utilizan los vehículos como nodos de comunicación y por lo tanto el consumo de éstos puede ser una variable interesante de estudio. En resumen, implementando el modelo de consumo eléctrico dentro de un simulador de redes tan importante como el ns-3 estamos ayudando a cualquier investigación que comprenda redes de telecomunicaciones y vehículos eléctricos.

6 ESTUDIO Y DISEÑO DEL MÓDULO

Como se pretende añadir funcionalidad a un sistema ya existente, la fase de diseño del nuevo módulo es completamente dependiente de la arquitectura del simulador. Por lo tanto, el primer trabajo consistirá en el estudio y análisis de cómo se organiza el código en ns-3 para entender el diseño que habrá que implementar.

Afortunadamente, el proyecto de ns-3 cuenta con una documentación muy amplia y relativamente actualizada. La documentación especifica que el simulador está compuesto por un *core* o núcleo, que implementa toda la infraestructura básica que se necesita. El resto se trata de módulos utilizados para cada tipo de simulación. Por ejemplo, tenemos un módulo para simular comunicaciones Wifi, otro para el protocolo IP, otro que se encarga de la movilidad de los nodos, y así con cualquier aspecto que se desee simular.

6.1 ENFOQUE DEL DISEÑO

Este trabajo consiste en la creación de un nuevo módulo que se encargue de simular el consumo de vehículos. Se habla del consumo de vehículos en general porque, aunque se vaya a implementar el consumo de vehículos eléctricos, se facilitará la posterior agregación del consumo de otro tipo de vehículos.

En general, cada módulo está compuesto por:

- ***model*:** Modelo de simulación que implementa todos los cálculos matemáticos.

[2] SUMO (del inglés *Simulation of Urban MObility*) es un simulador muy conocido para realizar simulaciones de tráfico urbano con el que se puede obtener también los consumos de combustible fósil o eléctrico [48].

- ***helper*:** Clase de ayuda que implementa todos los métodos necesarios para llevar a cabo una simulación (leer datos de un fichero, crear el modelo, etc.).

- ***example*:** Ejemplo de *script* que realiza una simulación de prueba para que el usuario vea cómo utilizar el código implementado.

Así pues, se ha utilizado esta estructura y se han seguido las indicaciones de la documentación [42], para crear un nuevo módulo pensando en las siguientes cuestiones: funcionalidad, reusabilidad y dependencias del código.

6.1.1 Funcionalidad

La funcionalidad del módulo será la siguiente:

- El usuario ejecutará el *script* de la simulación pasando como parámetros un fichero con las trazas de movilidad de cada nodo, un fichero XML con los atributos de cada modelo de vehículo eléctrico, el número de nodos que participan en la simulación, la duración de la simulación y el tiempo de actualización (es decir, cada cuánto se actualiza la energía del vehículo en la simulación).

- El formato de las trazas de movilidad de los nodos será el formato Tcl compatible con el ns-3.

- Cada vez que se actualice la energía consumida del vehículo, se generará un evento que podrá ser capturado por el usuario del *script* para tener acceso a todas las variables del modelo consumido y poder sacar estadísticas durante la simulación por la salida estándar o en un fichero.

- El diseño del fichero XML con los atributos de cada vehículo eléctrico estará definido de la siguiente manera:

```
<ElectricVehicles>
    <ElectricVehicle node="0">
        <!-- ELECTRIC BUS PARAMETERS -->
        <param key="initialEnergy" value="15000"/>
        <param key="maximumBatteryCapacity" value="324000"/>
        <param key="vehicleMass" value="19000"/>
        <param key="frontSurfaceArea" value="8.25"/>
        <param key="airDragCoefficient" value="0.6"/>
        <param key="internalMomentOfInertia" value="0.01"/>
        <param key="radialDragCoefficient" value="0.5"/>
        <param key="rollDragCoefficient" value="0.01"/>
        <param key="constantPowerIntake" value="100"/>
        <param key="propulsionEfficiency" value="0.9"/>
        <param key="recuperationEfficiency" value="0.9"/>
    </ElectricVehicle>
    <ElectricVehicle node="1">
                                    ...
                                    ...
                                    ...
    </ElectricVehicle>
</ElectricVehicles>
```

6.1.2 Reusabilidad

La documentación del ns-3 recomienda que, antes de empezar con la implementación de un nuevo módulo, se reflexione sobre la reusabilidad del diseño. Es decir, pensar en que los usuarios deberán de poder integrar el modelo de consumo con otros modelos, para realizar toda clase de simulaciones según lo que deseen investigar. Para ello, es imprescindible pensar en un buen diseño de la interfaz de programación de aplicaciones o API (del inglés *Application Programming Interface*) para que el módulo desarrollado sea fácilmente utilizado por terceros.

Con el objetivo de hacer muy fácil el uso del modelo de consumo, se ha diseñado la API de tal manera que con crear el objeto de la clase *helper* y hacer una llamada a una función de "instalación" o "puesta en marcha", se realice la simulación del consumo de cada nodo. De esta forma, un ejemplo de uso muy simple del módulo sería el siguiente:

1. Crear la clase de ayuda para la simulación

```
ElectricConsumptionHelper electricConsumption =
        ElectricConsumptionHelper (vehicleAttributesFile, updateTime);
```

Fichero XML con los atributos de cada vehículo

Decimal con el tiempo de actualización

2. "Instalar" el modelo de consumo en cada nodo

```
electricConsumption.Install ();
```

Figura 3 - Uso del módulo implementado

Solamente con estas dos líneas de código abstraemos al usuario de toda complejidad sobre el funcionamiento interno y facilitamos la reusabilidad del módulo. Por otro lado, el diseño del diagrama de clases ha sido planteado para facilitar la agregación de otros modelos de consumo (gasolina, gasoil, etc.) mediante una clase abstracta genérica, como se mostrará posteriormente.

6.1.3 Dependencias

Minimizar al máximo las dependencias externas del módulo con otros es importante para la modularidad del código y para su mantenimiento. A pesar de ello, pueden existir dependencias inevitables, dadas las necesidades del módulo a implementar. En este caso, la única dependencia encontrada que no se puede esquivar es la utilización del módulo de movilidad (en el código *mobility model*). Esto es lógico, puesto que el cálculo del consumo de un vehículo siempre va a depender de su movilidad y velocidad.

A pesar de esto, no se puede considerar una dependencia crítica, puesto que es un módulo muy consolidado en el código del ns-3, y que apenas sufre cambios en las nuevas versiones.

6.2 DIAGRAMA DE CLASES

A continuación, se muestra el diagrama de clases diseñado para realizar la implementación del módulo de consumo de vehículos. Primero de manera más esquemática, y posteriormente de forma más detallada para cada clase.

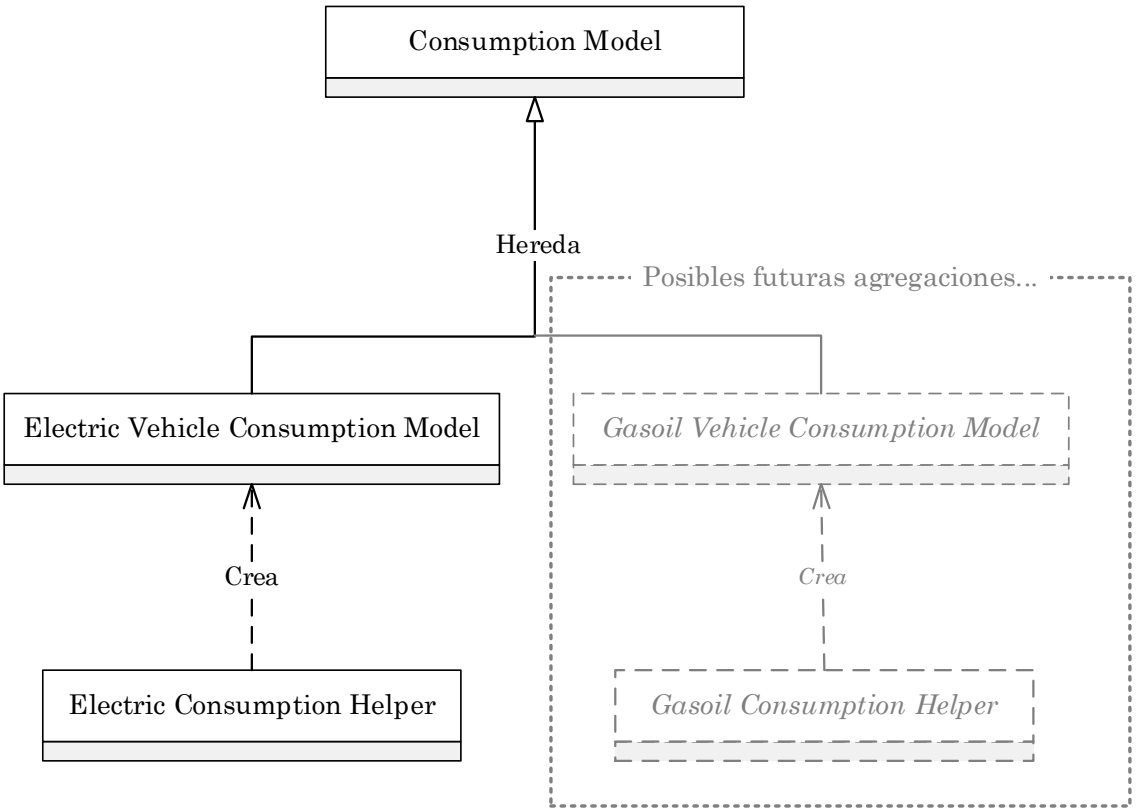


Figura 4 - Esquema básico del diagrama de clases

Como se puede observar, la clase *ElectricConsumptionHelper* es la encargada de crear los modelos de consumo eléctrico para cada nodo (*ElectricVehicleConsumptionModel*), que a su vez éstos heredan de la clase abstracta de modelo de consumo (*ConsumptionModel*). Este diseño permite, como se contempla en el diagrama, añadir de forma sencilla nuevos modelos de consumo según el tipo de vehículo.

6.2.1 *ConsumptionModel*

Consumption Model <<*Abstract*>>
- MobilityModel: *Ptr<MobilityModel>* - LastPosition: *Vector* - LastVelocity: *Vector* - LastUpdateTime: *Time*
Public Functions - static *TypeId* GetTypeId () - ConsumptionModel () - ~ ConsumptionModel () - *virtual void* UpdateConsumption ()

Figura 5 - Modelo UML de la clase ConsumptionModel

Esta clase abstracta agrupa todos los atributos que tienen en común cualquier modelo de consumo de un vehículo: movilidad, posición y velocidad en el último momento, y tiempo. Por otro lado, la clase contiene el método

abstracto `UpdateConsumption()` que deberá implementar cada clase heredera, especificando los cálculos según el tipo de consumo. También contiene los métodos constructores y destructores, así como un método `GetTypeId()` que utiliza el simulador internamente para registrar el objeto en la simulación.

6.2.2 *ElectricVehicleConsumption*

Heredando de *ConsumptionModel*, esta clase implementa el modelo de consumo de un vehículo eléctrico. Como se aprecia en el diseño UML de la Figura 6, la clase contiene todos los atributos necesarios para los cálculos especificados en la sección de Modelo de consumo eléctrico. El atributo *RemainingEnergyWh* es del tipo *TracedValue<double>* propio del ns-3, que nos permite generar un evento en la simulación cada vez que la energía del vehículo se modifica. La clase contiene también todos los métodos privados necesarios para realizar los cálculos del consumo: distancia entre posiciones, obtención de ángulos, diferencias de energía, etc...

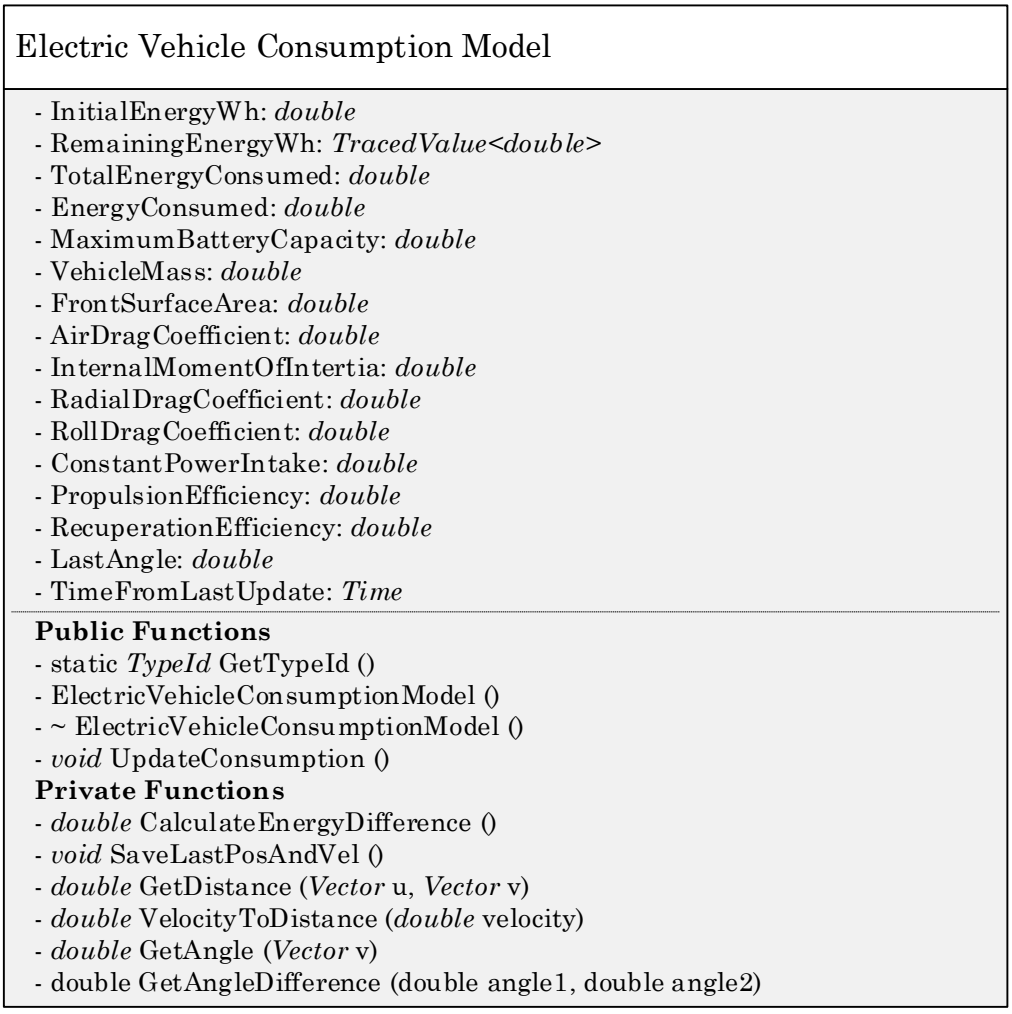


Figura 6 - Modelo UML de la clase ElectricVehicleConsumptionModel

6.2.3 *ElectricConsumptionHelper*

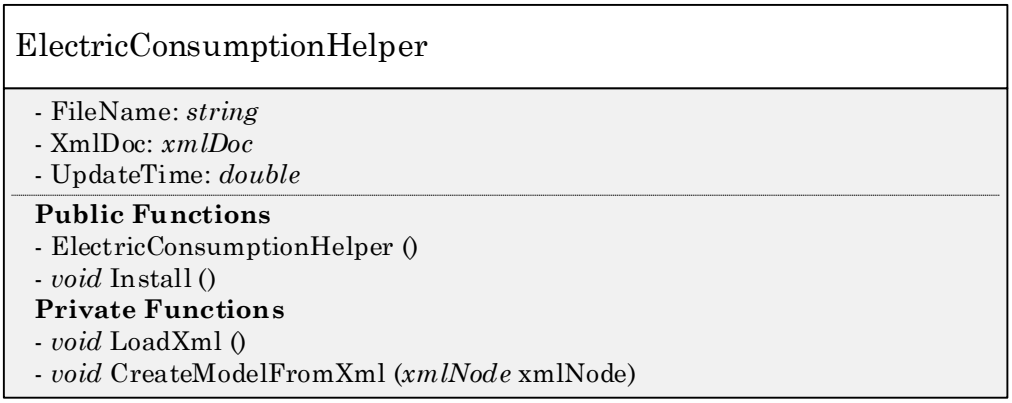


Figura 7 - Modelo UML de la clase ElectricConsumptionHelper

La clase *ElectricConsumptionHelper* se utiliza para crear el modelo de consumo de vehículo eléctrico según los atributos especificados en un fichero .XML. Dispone de los atributos necesarios para ello con el nombre del fichero, un objeto del documento XML utilizado para leerlo, y el valor decimal de actualización para especificar cada cuánto tiempo se actualiza el consumo. La clase define el método público `Install()` que realiza todo el trabajo de puesta en marcha de los modelos de consumo, haciendo uso para ello de los métodos privados.

7 IMPLEMENTACIÓN

A continuación, se explican varios aspectos que se han tenido en cuenta al realizar la implementación del proyecto. Los ficheros de código resultantes han sido adjuntados al final del documento como anexo.

7.1 CONTROL DE VERSIONES

La gestión del código implementado se ha realizado mediante el sistema de control de versiones *git*. El repositorio oficial del ns-3 funciona con el sistema de control de versiones *Mercurial* pero existe una copia oficial del repositorio en *GitHub* (https://github.com/nsnam/ns-3-dev-git) al que se le ha hecho un *fork*[3] para añadir el código implementado. Se ha decidido utilizar la forja de *GitHub* al tener más experiencia con *git* y tratarse de una plataforma bastante más conocida para el desarrollo de proyectos *open-source*. El *fork* creado con el código implementado puede encontrarse en la siguiente dirección: https://github.com/ssalvatella/ns-3-mobility-consumption

[3] Una bifurcación (en inglés *fork*) es la creación de un proyecto en una dirección distinta de la principal u oficial, tomando el código fuente del proyecto ya existente.

7.2 ESTILO DEL CÓDIGO

Para realizar la implementación del diseño se ha tenido que utilizar el estilo de código especificado por la documentación del ns-3 [43]. En ella, se indica que hay que utilizar el estándar de código GNU así como las siguientes convenciones:

- Uso de la convención *CamelCase* para los nombres funciones. Las palabras se escriben juntas sin espacios y con la primera letra en mayúsculas. Por ejemplo, "my computer" se transformaría a "MyComputer".
- Los nombres de variables utilizan también *CamelCase* pero con la primera letra en minúscula.
- Las variables globales utilizan el prefijo "*g_*" y las variables de atributos de clases el prefijo "*m_*".
- Los nombres de constantes se escriben enteramente en mayúsculas.
- Utilizar el inglés obligatoriamente para todo el código.
- Uso de sustantivos para nombres de variables y de verbos para nombres de funciones.

Para la documentación del código se ha utilizado el generador Doxygen, por lo que los comentarios están adaptados al formato que utiliza.

7.3 ASPECTOS CONCRETOS DEL NS-3

En este apartado se procede a explicar en rasgos generales aspectos más concretos que son propios del simulador ns-3 y que hay que tener en cuenta para la implementación.

El simulador ns-3 está construido de tal manera que para implementar la simulación del consumo de vehículos hay que primero añadir a cada nodo su movilidad asociada y posteriormente el modelo de consumo del vehículo. De esta manera, cada nodo contiene en su interior un modelo de movilidad y un modelo de consumo del vehículo.

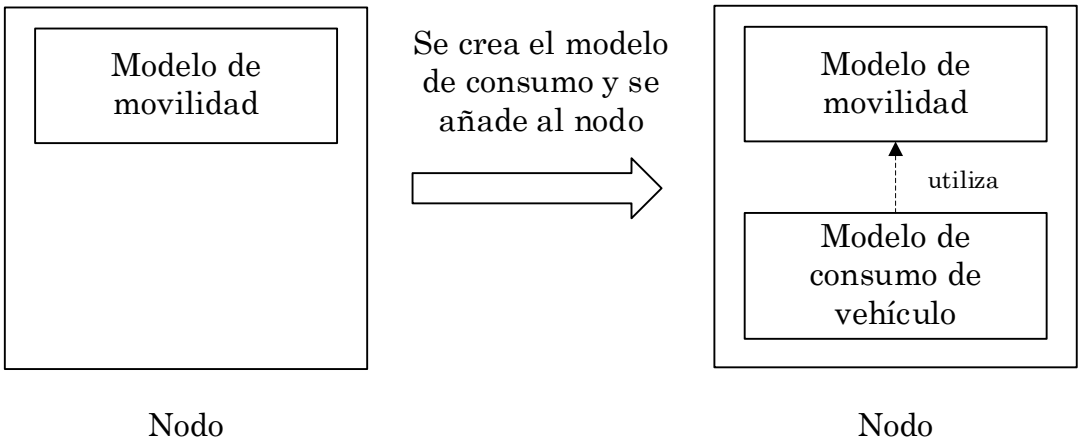


Figura 8 - Nodo que contiene distintos objetos para la simulación

El *core* del simulador nos aporta herramientas para facilitar la creación de objetos y gestionar fácilmente los punteros a memoria. De esta manera, el modelo de consumo eléctrico lo creamos con la función:

```
Ptr<ElectricConsumptionModel> model =
                CreateObject<ElectricConsumptionModel> ();
```

Nótese que el resultado de la creación del objeto es del tipo `Ptr<?>`*, tipo propio del core del simulador que gestiona automáticamente la memoria de los recursos utilizados.*

Una vez con el modelo creado se añade al nodo mediante el método:

```
node->AggregateObject (model);
```

Otro aspecto a tener en cuenta en la implementación es la generación de eventos para que durante la simulación se puedan ir registrando estadísticas sobre el consumo de los vehículos. Para ello, la variable que representa la energía del vehículo ha sido especificada con el tipo de `TracedValue<double>` para que el simulador genere un evento cada vez que se modifique su valor. Para capturar el evento y ejecutar un método definido por el usuario en el script se deberá de utilizar el método propio del ns-3:

```
Config::Connect(
"/NodeList/*/$ns3::ElectricVehicleConsumptionModel/RemainingEnergy",
MakeCallback (&MetodoDefinido)
);
```

El primer parámetro es una cadena que representa una dirección que señala a un atributo perteneciente a cualquier nodo de la simulación (“`/NodeList/*`”) en el objeto de tipo *ElectricVehicleConsumptionModel* con el nombre *RemainingEnergy*. Con este parámetro especificamos a qué atributo queremos escuchar sus cambios. El segundo parámetro es la función que se ejecutará cuando se capture un evento.

8 RESULTADOS

Una vez que la implementación del modelo está lista, es pertinente realizar varias simulaciones con las que obtener una serie de datos para validar posteriormente que el modelo funciona correctamente. Para validar los datos obtenidos de las simulaciones, en un primer momento, se ha comprobado que sean valores con sentido y tengan cierta lógica respecto al escenario simulado. Después, se ha ejecutado la misma simulación dentro del simulador de movilidad urbana SUMO y se han obtenido los datos de consumo según su modelo implementado. El objetivo de esto es comparar los datos obtenidos por SUMO con los del modelo implementado en este trabajo dentro del ns-3.

A continuación, se explican los resultados de las dos simulaciones realizadas, una simulación sencilla y otra bastante más compleja.

8.1 SIMULACIÓN DE VEHÍCULO EN LÍNEA RECTA

Para la primera simulación de prueba, se ha ideado un escenario extremadamente simple. De esta forma, se ha podido comprobar a simple vista que los datos obtenidos tienen sentido.

Mediante la herramienta NETEDIT incluida en el *software* de SUMO se ha ideado un escenario en el que un vehículo está parado, arranca, y alcanza una velocidad máxima de 50 Km/h que se mantiene hasta llegar a los 500 m recorridos.

Figura 9 - Vehículo simulado en SUMO recorriendo una línea recta

Después, se han generado las trazas del recorrido del vehículo y convertidas al formato TCL (mediante un *script* incluido en SUMO) para que el simulador ns-3 pueda leer la movilidad del vehículo.

Una vez con la movilidad lista para que la pueda leer el modelo implementado en el **ns-3** se ejecuta la simulación estableciendo los siguientes parámetros del vehículo:

Capacidad máxima de la batería	Masa del vehículo	Superficie del área frontal	Coeficiente de rozamiento del aire	Momento de inercia interno
324 kWh	**18 000 kg**	**3,25 m²**	**0,6**	**0,01 kg · m²**

Coeficiente de rozamiento radial	Coeficiente de rozamiento de rodamiento	Consumo constante	Eficiencia de propulsión	Eficiencia de recuperación
0,5	**0,01**	**100 Wh**	**0,9**	**0,9**

Tabla 3 - Atributos del vehículo para la simulación

Los parámetros del vehículo para la simulación han sido escogidos para simular el modelo de autobús de 10,8 metros eléctrico fabricado por la empresa BYD [44].

Los resultados de la simulación, incluyendo los obtenidos por SUMO para su comparación[4] son los siguientes:

Tiempo (s)	Velocidad (m/s)	Energía ns-3 (Wh)	Energía SUMO (Wh)	Diferencia energía	Error cuadrático
1	1	3,53926	3,5392	-0,00006	0,00000
2	2	9,98469	9,9851	0,00041	0,00000
3	3	16,4415	16,442	0,00050	0,00000
4	4	22,9148	22,9156	0,00080	0,00000
5	5	29,4102	29,4111	0,00090	0,00000
6	6	35,9331	35,9343	0,00120	0,00000
7	7	42,4892	42,4906	0,00140	0,00000
8	8	49,084	49,0856	0,00160	0,00000
9	9	55,7229	55,7246	0,00170	0,00000
10	10	62,4114	62,4134	0,00200	0,00000
11	11	69,1551	69,1573	0,00220	0,00000
12	12	75,9596	75,962	0,00240	0,00001
13	13	82,8303	82,8328	0,00250	0,00001
14	13,9	81,4809	81,4836	0,00270	0,00001

[4] Para obtener rápidamente los datos del consumo según SUMO se ha desarrollado un pequeño *script* en Python que lee el fichero de salida de SUMO y lo escribe en otro con un formato compatible con Excel.

15	13,9	10,4947	10,4975	0,00280	0,00001
16	13,9	10,4947	10,4975	0,00280	0,00001
17	13,9	10,4947	10,4975	0,00280	0,00001
18	13,9	10,4947	10,4975	0,00280	0,00001
19	13,9	10,4947	10,4975	0,00280	0,00001
20	13,9	10,4947	10,4975	0,00280	0,00001
21	13,9	10,4947	10,4975	0,00280	0,00001
22	13,9	10,4947	10,4975	0,00280	0,00001
23	13,9	10,4947	10,4975	0,00280	0,00001
24	13,9	10,4947	10,4975	0,00280	0,00001
25	13,9	10,4947	10,4975	0,00280	0,00001
26	13,9	10,4947	10,4975	0,00280	0,00001
27	13,9	10,4947	10,4975	0,00280	0,00001
28	13,9	10,4947	10,4975	0,00280	0,00001
29	13,9	10,4947	10,4975	0,00280	0,00001
30	13,9	10,4947	10,4975	0,00280	0,00001
31	13,9	10,4947	10,4975	0,00280	0,00001
32	13,9	10,4947	10,4975	0,00280	0,00001
33	13,9	10,4947	10,4975	0,00280	0,00001
34	13,9	10,4947	10,4975	0,00280	0,00001
35	13,9	10,4947	10,4975	0,00280	0,00001
36	13,9	10,4947	10,4975	0,00280	0,00001
37	13,9	10,4947	10,4975	0,00280	0,00001
38	13,9	10,4947	10,4975	0,00280	0,00001
39	13,9	10,4947	10,4975	0,00280	0,00001
40	13,9	10,4947	10,4975	0,00280	0,00001
41	13,9	10,49470	10,4975	0,00280	0,00001
	Total:	0,920 kWh	0,920 kWh	0,09585 Wh	
Error cuadrático medio (EMC):			0,00001 Wh		

Tabla 4 - Resultados de la simulación de consumo eléctrico en línea recta

Lo primero que se puede observar en los resultados obtenidos es que la diferencia entre el cálculo del modelo de consumo implementado en el ns-3 y el disponible en SUMO es prácticamente mínimo. De hecho, la diferencia entre el total del consumo calculado por SUMO y el calculado por el modelo implementado es del 0,0104%. Esta diferencia mínima se traduce en que el error cuadrático medio utilizando como referencia los valores de SUMO este estimado en 0,00001 Wh. Por lo tanto, se puede afirmar que obtenemos valores muy similares a los obtenidos por otro modelo de confianza y utilizado ampliamente.

Para comprobar que los valores obtenidos son valores lógicos con la naturaleza del escenario simulado, se disponen de las siguientes gráficas que

muestran la evolución del consumo de energía y la velocidad del vehículo durante el recorrido:

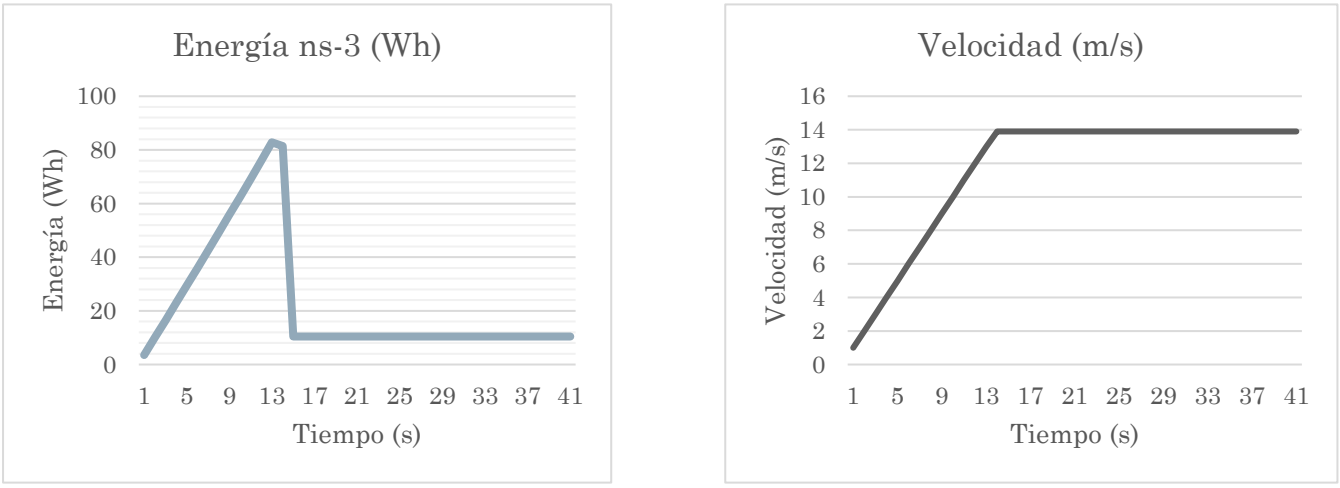


Figura 10 - Consumo de energía y velocidad del vehículo durante la simulación

Observando las gráficas obtenidas se puede constatar que los valores obtenidos son físicamente lógicos. Cuando el vehículo arranca y acelera hasta llegar una velocidad constante de 14 m/s (50 Km/h), la energía consumida aumenta progresivamente. Una vez que el vehículo alcanza su velocidad de crucero, el consumo de la energía desciende radicalmente y se mantiene estable al no tener que aumentar su energía cinética y solamente consumir la energía perdida por el rozamiento.

La siguiente simulación realizada presenta una mayor complejidad y duración, pero con ella se han obtenido datos interesantes.

8.2 SIMULACIÓN DE UNA LÍNEA DE AUTOBÚS

Después de realizar una simulación muy sencilla, se ha ideado un nuevo escenario, pero esta vez con una aplicación real. El escenario escogido ha sido simular una línea de bus urbano de la ciudad de Teruel y realizar el cálculo con el modelo implementado en el ns-3.

La fase de preparación para la generación de las trazas del vehículo es bastante más compleja en este caso. La línea de autobús escogida ha sido la línea B que atraviesa los barrios de la Fuenfresca, Ensanche y el centro de la ciudad [45]. Para poder realizar esta simulación hace falta construir en el simulador SUMO un mapa real de la ciudad de Teruel. Para ello, se ha exportado el mapa de la ciudad desde la plataforma de OpenStreetMap obteniendo un fichero de coordenadas OSM.

Una vez con el mapa, hay que poder manipularlo y utilizarlo en el *software* de SUMO. Para esta tarea, SUMO dispone de una herramienta que nos permite transformar mapas OSM a un formato reconocible por SUMO. La generación del mapa en la interfaz de SUMO puede contener errores en las

carreteras debido a que los datos de OpenStreetMap no siempre son muy precisos, especialmente en ciudades pequeñas como Teruel. Para corregir estos errores, se ha hecho uso nuevamente de la herramienta NETEDIT retocando cualquier carretera que pudiera dar problemas, consiguiendo un resultado final bastante satisfactorio como se puede ver en la Figura 11.

Figura 11 - Mapa de Teruel generado dentro de SUMO

Con el mapa listo, lo siguiente es definir la ruta del bus para la simulación. Para ello, se ha consultado el trayecto en la web y se ha definido en el archivo de rutas que utiliza SUMO con cada calle del recorrido. Para cada parada de la línea se ha establecido un tiempo medio de 30 segundos.

Después de comprobar que el trayecto se simulaba correctamente en SUMO, se han generado las trazas del viaje para poder pasarlas posteriormente en el modelo de consumo implementado en este trabajo.

Para el vehículo simulado, se han utilizado los mismos atributos descritos anteriormente en la Tabla 2. El objetivo de esta simulación es ver, como ejemplo práctico, qué consumo eléctrico supondría cambiar los autobuses actuales de gasoil por autobuses eléctricos.

A continuación, se muestran los resultados de la simulación. Como el trayecto del bus dura 27 minutos y medio, se muestran los totales de la simulación, en vez de los valores en cada instante.

	Tiempo	Energía consumida según **ns-3**	Energía consumida según SUMO	Diferencia de energía	Error cuadrático medio
Total:	27 min 48 s	9,261 kWh	10,509 kWh	1,247 kWh	289 Wh

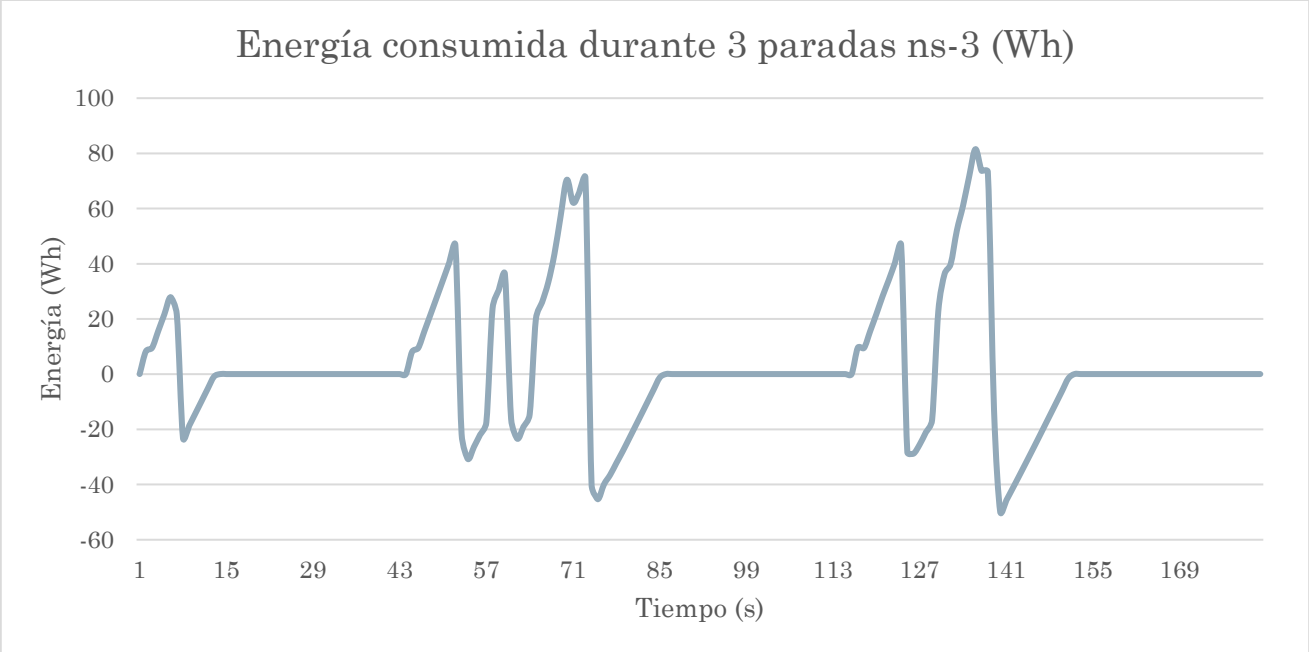
Tabla 5 - Resultados finales de la simulación de la línea de bus

Como se puede observar, para una simulación tan larga de casi media hora, la diferencia entre la energía consumida obtenida por SUMO y la del modelo de consumo implementado es mayor que en la anterior simulación. Esto es normal si pensamos que la anterior simulación solamente era de unos segundos comparada con esta mucho más larga.

El error cuadrático medio se sigue manteniendo bastante bajo teniendo en cuenta que 289 Wh para un vehículo eléctrico del tamaño de un autobús de 18 toneladas es bastante poco.

En lo que respecta a la energía consumida, el resultado del modelo implementado nos muestra que por un trayecto de la línea B el autobús gastaría en torno al 12% de su batería. Eso significa que, dejando un margen para evitar complicaciones, el autobús podría hacer en torno a seis trayectos sin tener que volverse a cargar.

Analizar todo el consumo del trayecto entero en una gráfica es bastante confuso al haber demasiados datos en poco espacio. Por ello, se disponen a continuación, las gráficas del consumo y la velocidad durante tres paradas de bus para entender como evoluciona el gasto de energía en función del trayecto.



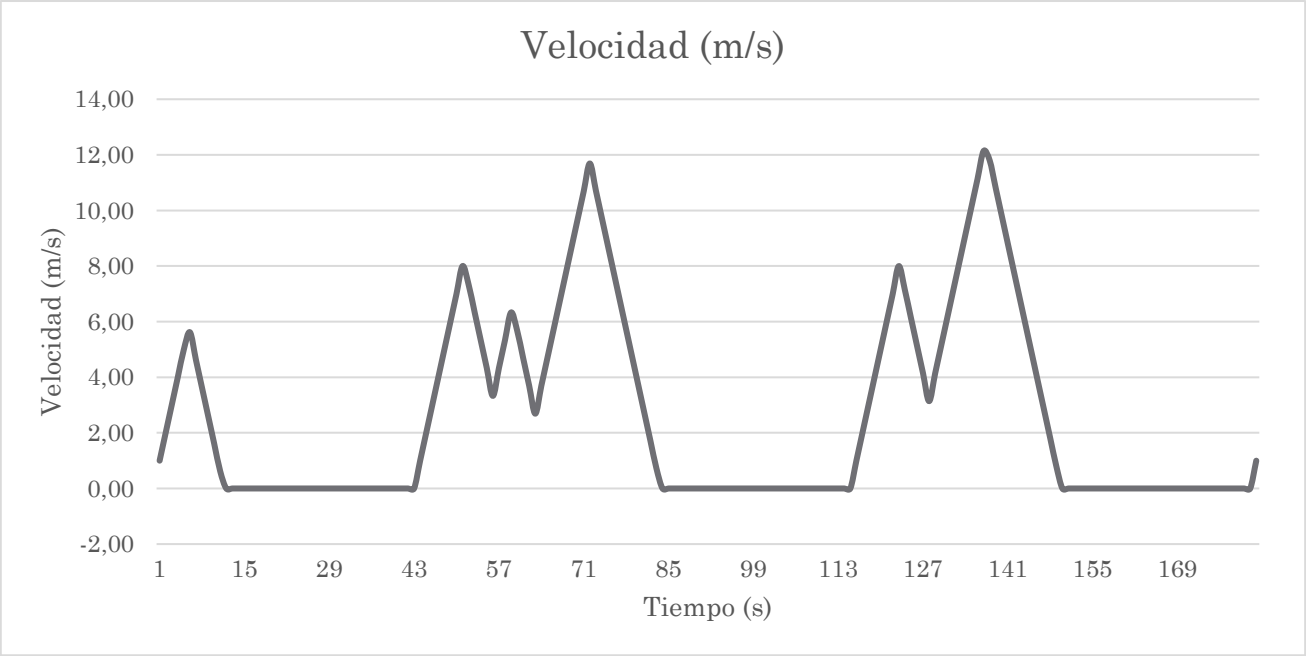


Figura 12 - Evolución del consumo y la velocidad durante el trayecto de bus

Lo primero que puede llamar la atención son los valores negativos en el consumo de energía. Esto es debido a que el vehículo eléctrico puede obtener energía al transformar la energía cinética en energía eléctrica durante el frenado. Podemos confirmarlo viendo que cuando la velocidad desciende, el consumo de energía se vuelve negativo. Esto nos confirma la importancia que tiene para un vehículo eléctrico el aprovechamiento que logra de la energía en el frenado. De hecho, durante toda la simulación el vehículo adquiere un total de 13,748 kWh gracias al frenado, ahorrando así energía al consumo total.

Es interesante también ver, comparando la simulación anterior con ésta, cómo mantener una velocidad constante es muy importante para un vehículo eléctrico. En la primera simulación se puede observar que mantener una velocidad constante reduce considerablemente el consumo eléctrico mientras que, en la segunda simulación, al haber muchas paradas y arranques, encontramos numerosos picos de consumo. Crear carriles especiales para el transporte público que hagan más fluido su recorrido adquiere una mayor importancia en caso de que el transporte sea eléctrico.

A modo de curiosidad, con este tipo de simulaciones se pueden hacer estimaciones del gasto que supondría el consumo de estos vehículos en una línea concreta. En este caso, suponiendo una tarifa eléctrica común y sabiendo que el precio medio del kilovatio hora en España es de 0,147 €/kWh, el coste total de un trayecto de la línea sería de 1,36 €. Es decir, que con que solo dos personas se subieran durante el trayecto y pagaran un billete sencillo de un euro ya se amortizaría el consumo del vehículo.

9 CONCLUSIONES Y PERSPECTIVAS FUTURAS

El modelo de consumo de vehículos eléctricos implementado dentro del ns-3 puede utilizarse con una precisión adecuada en multitud de escenarios relacionados con el consumo de energía de vehículos y su optimización. Además, podrá ser utilizado para simulaciones que impliquen comunicaciones entre vehículos (VANET). La implementación realizada también permitirá en el futuro la agregación de nuevos modelos de consumo para otro tipo de vehículos como gasoil, gasolina o gas natural.

Los objetivos propuestos al inicio del trabajo se han cumplido. Se ha analizado y estudiado la situación actual del vehículo eléctrico. Después, se ha implementado el modelo de consumo de vehículos eléctricos con éxito dentro del simulador *open-source* ns-3. Posteriormente, se ha comprobado la validez y el correcto funcionamiento de lo implementado analizando varias simulaciones. Por último, el código desarrollado está pendiente de ser aprobado para la distribución oficial del ns-3.

Este trabajo en sí conforma un punto de partida. A partir de lo realizado, el trabajo puede continuar en las siguientes líneas:

1. Implementar los modelos de consumo para todos los tipos de combustibles más utilizados (gasoil, gasolina, o gas natural) aprovechando para ello el diseño de clases realizado.

2. Desarrollar algoritmos mediante técnicas de Inteligencia Artificial que optimicen el consumo de los vehículos mediante comunicaciones entre ellos.

3. Realizar mediciones reales en campo con vehículos eléctricos para compararlas con los valores obtenidos por el modelo implementado y hacer cualquier corrección que fuera pertinente para mejorar aún más la precisión del modelo.

10 REFERENCIAS

[1] J. L. Powell, «Climate Scientists Virtually Unanimous,» *Bulletin of Science, Technology & Society,* vol. 35, nº 5, pp. 121-124, 2016.

[2] Agencia Internacional de la Energía, «Emisiones de CO2 originadas por el transporte (% del total de la quema de combustible),» IEA, 1 Enero 2014. [En línea]. Disponible en: https://datos.bancomundial.org/indicador/EN.CO2.TRAN.ZS [Último acceso: 14 de junio 2018].

[3] M. Guarnieri, «Looking Back to Electric Cars,» de *History of Electro-technology Conference*, Pavia, 2012.

[4] J. Larminie y J. Lowry, Electric Vehicle Technology Explained, Chichester: John Wiley & Sons Ltd, 2012.

[5] N. Burton, A History of Electric Cars, Ramsbury, Marlborough: The Crowood Press Ltd, 2013.

[6] J. Y. Yong, V. K. Ramachandaramurthy, K. M. Tan y N. Mithulananthan, «A review on the state-of-the-art technologies of electric vehicle, its impacts and prospects,» *Renewable and Sustainable Energy Reviews,* nº 49, pp. 365-385, 2015.

[7] International Energy Agency, Global EV Outlook: Understanding the Electric Vehicle Landscape to 2020, 2013.

[8] International Energy Agency, Tracking Clean Energy Progress 2017, 2017.

[9] International Energy Agency, Global EV Outlook: Two million and counting, Paris: OECD/IEA, 2017.

[10] «World's Top 10 Best-Selling Electric Vehicles,» Gear Heads, 2017. [En línea]. Disponible en: https://gearheads.org/worlds-top-10-best-selling-electric-vehicles/ [Último acceso: 14 de junio 2018].

[11] Clean Energy Ministerial, «Electric Vehicles Initiative (EVI),» Clean Energy Ministerial, Junio 2016. [En línea]. Disponible en: http://www.cleanenergyministerial.org/Our-Work/Initiatives/Electric-Vehicles [Último acceso: 14 de junio 2018].

[12] Clean Energy Ministerial, «Electric Vehicles Initiative (EVI) - Fact sheet,» Junio 2016. [En línea].

Disponible en:
http://www.cleanenergyministerial.org/Portals/2/pdfs/factsheets/EVI-CEM7-FS.pdf
[Último acceso: 14 de junio 2018].

[13] International Energy Agency, Global EV Outlook: Beyond one million electric cars, Paris: OECD/IEA, 2016.

[14] ANFAC, «Informe Anual,» 2016.

[15] ACEA, «Interactive map: Correlation between uptake of electric cars and GDP in the EU,» European Automobile Manufacturers Association, 31 Octubre 2017. [En línea].
Disponible en:
http://www.acea.be/statistics/article/interactive-map-correlation-between-uptake-of-electric-cars-and-gdp-in-EU
[Último acceso: 14 de junio 2018].

[16] ANFAC, «Informe Anual ANFAC 2016,» Asociación española de Fabricantes de Automóviles y Camiones, 2016. [En línea].
Disponible en:
http://www.anfac.com/memoria/memoriaAnfac2016.htm
[Último acceso: 14 de junio 2018].

[17] ElectroMaps, «Puntos de recarga en España,» ElectroMaps, 2017. [En línea].
Disponible en:
https://www.electromaps.com/puntos-de-recarga/espana
[Último acceso: 14 de junio 2018].

[18] S. Amadoz, «La odisea del coche eléctrico: pocos puntos de recarga y mal repartidos,» El Motor, 18 Febrero 2017. [En línea].
Disponible en:
https://motor.elpais.com/electricos/puntos-de-recarga-coches-electricos/
[Último acceso: 14 de junio 2018].

[19] Boletín Oficial del Estado (BOE), «Real Decreto-ley 9/2013, de 12 de julio, por el que se adoptan medidas urgentes para garantizar la estabilidad financiera del sistema eléctrico,» 13 Julio 2013. [En línea].
Disponible en:
http://www.boe.es/boe/dias/2013/07/13/pdfs/BOE-A-2013-7705.pdf
[Último acceso: 14 de junio 2018].

[20] Boletín Oficial del Estado (BOE), «Real Decreto 1078/2015, de 27 de noviembre, por el que se regula la concesión directa de ayudas para la adquisición de vehículos de energías alternativas, y para la implantación de puntos de recarga de vehículos eléctricos en 2016, MOVEA,» 28 Noviembre 2015. [En línea].

Disponible en:
https://www.boe.es/boe/dias/2015/11/28/pdfs/BOE-A-2015-12900.pdf
[Último acceso: 14 de junio 2018].

[21] Boletín Oficial del Estado (BOE), «Real Decreto 617/2017, de 16 de junio, por el que se regula la concesión directa de ayudas para la adquisición de vehículos de energías alternativas, y para la implantación de puntos de recarga de vehículos eléctricos en 2017 (Plan MOVEA 2017).,» 23 Junio 2017. [En línea].
Disponible en:
https://www.boe.es/boe/dias/2017/06/23/pdfs/BOE-A-2017-7165.pdf
[Último acceso: 14 de junio 2018].

[22] Europa Press, «Así es el Plan ProMovea que se aprobará en las próximas semanas para incentivar el mercado de coches eléctricos,» Ecomotor.es, 27 Octubre 2017. [En línea].
Disponible en:
http://www.eleconomista.es/ecomotor/motor/noticias/8704471/10/17/Asi-es-el-Plan-ProMovea-que-se-aprobara-en-las-proximas-semanas-para-incentivar-el-mercado-de-coches-electricos.html
[Último acceso: 14 de junio 2018].

[23] A. Lenormand, «Transition écologique : un budget en hausse mais recentré sur les priorités du Plan climat,» Caisse des Dépôts, 28 Septiembre 2017. [En línea].
Disponible en:
https://www.caissedesdepotsdesterritoires.fr/cs/ContentServer?pagename=Territoires/Articles/Articles&cid=1250279794928
[Último acceso: 14 de junio 2018].

[24] movilidadelectrica.com, «Arranca el plan de ayudas para vehículos eléctricos en Alemania,» movilidadelectrica.com, 3 Julio 2016. [En línea].
Disponible en:
https://movilidadelectrica.com/arranca-plan-ayudas-vehiculos-electricos-alemania/
[Último acceso: 14 de junio 2018].

[25] ANFAC, «ANFAC considera imprescindible la ampliación de fondos del Plan MOVALT para potenciar el vehículo alternativo,» 14 Diciembre 2017. [En línea].
Disponible en:
http://www.anfac.com/noticias.action?idDoc=13829&accion=noticias_anfac
[Último acceso: 14 de junio 2018].

[26] E. Schmidt, «The impact of growing electric vehicle adoption on electric utility grids,» FleetCarma, 28 Agosto 2017. [En línea].
Disponible en:

https://www.fleetcarma.com/impact-growing-electric-vehicle-adoption-electric-utility-grids/

[Último acceso: 14 de junio 2018].

[27] Coinc Blog, «¿Cuál es el consumo medio de electricidad en España?,» Coinc Blog, 30 Agosto 2015. [En línea].
Disponible en:
https://www.coinc.es/blog/noticia/consumo-medio-electricidad-hogar#
[Último acceso: 14 de junio 2018].

[28] International Energy Agency (IEA), Electricity Information 2017, 2017.

[29] International Energy Agency (IEA), CO2 emissions from fuel combustion, 2017.

[30] E. Ancillotti, R. Bruno y M. Conti, «The Role of Communication Systems in Smart Grids: Architectures, Technical Solutions and Research Challenges,» *Computer Communications,* p. 43, 2013.

[31] I. Cowie, «All About Batteries, Part 3: Lead-Acid Batteries,» EETimes, 13 Enero 2014. [En línea].
Disponible en:
https://www.eetimes.com/author.asp?section_id=36&doc_id=1320644
[Último acceso: 14 de junio 2018].

[32] The Committee on Climate Change, «Cost and performance of EV batteries,» 21 Marzo 2012. [En línea].
Disponible en:
https://www.theccc.org.uk/archive/aws/IA&S/CCC%20battery%20cost_%20Element%20Energy%20report_March2012_Public.pdf
[Último acceso: 14 de junio 2018].

[33] K. Tweed, «Tesla's Lithium-Ion Battery Catches Fire,» IEEE Spectrum, 3 Octubre 2013. [En línea].
Disponible en:
https://www.spectrum.ieee.org/energywise/transportation/advanced-cars/teslas-lithiumion-battery-catches-fire-
[Último acceso: 14 de junio 2018].

[34] A. Foley, B. Ó. Gallachóir y I. Winning, «State-of-the-Art in Electric Vehicle Charging Infrastructure,» *IEEE VPPC 2010: Vehicle Power and Propulsion Conference,* pp. 1-6, 2010.

[35] SAE, «SAE Charging Configurations and Ratings Terminology,» 2011. [En línea].
Disponible en:
http://www.sae.org/smartgrid/chargingspeeds.pdf
[Último acceso: 14 de junio 2018].

[36] Electric Light & Power, «IEC publishes two international EV standards,» Electric Light & Power, 19 Octubre 2011. [En línea].

Disponible en:
http://www.elp.com/articles/2011/10/iec-publishes-two-international-ev-standards.html
[Último acceso: 14 de junio 2018].

[37] CHAdeMO, «CHAdeMO Brochure Long,» 09 Septiembre 2013. [En línea].
Disponible en:
http://www.chademo.com/wp/wp-content/uploads/2013/09/20130925_brochure_long.pdf
[Último acceso: 14 de junio 2018].

[38] T. Kurczveil, P. Á. López y E. Schnieder, «Implementation of an Energy Model and a Charging Infraestructure in SUMO,» *Springer,* nº 8594, pp. 33-43, 2014.

[39] S. Keshav, «REAL: A Network Simulator,» *EECS Department, University of California, Berkeley,* nº UCB/CSD-88-472, 1988.

[40] T. Henderson, «[Ns-developers] announcing a new project for ns-3,» 3 Julio 2006. [En línea].
Disponible en:
http://mailman.isi.edu/pipermail/ns-developers/2006-July/002316.html
[Último acceso: 14 de junio 2018].

[41] Ns-3 Consortium, «Ns-3 Consortium Establishment Agreement,» 26 Abril 2013. [En línea].
Disponible en:
https://www.nsnam.org/wp-content/uploads/2013/01/NS-3-Consortium-Establishment-Agreement.pdf
[Último acceso: 14 de junio 2018].

[42] Ns-3, «Creating a new ns-3 model,» Ns-3, [En línea].
Disponible en:
https://www.nsnam.org/docs/manual/html/new-models.html
[Último acceso: 14 de junio 2018].

[43] Ns-3 Consortium, «Coding Style - ns-3,» ns-3, [En línea].
Disponible en:
https://www.nsnam.org/developers/contributing-code/coding-style/
[Último acceso: 14 de junio 2018].

[44] BYD, «BYD Auto, Build Your Dream,» BYD, 2018. [En línea].
Disponible en:
http://bydeurope.com/vehicles/ebus/types/10_8.php
[Último acceso: 06 Junio 2018].

[45] Samar, «Urbanos Teruel,» Samar, 2018. [En línea]. [Último acceso: 14 de junio 2018].

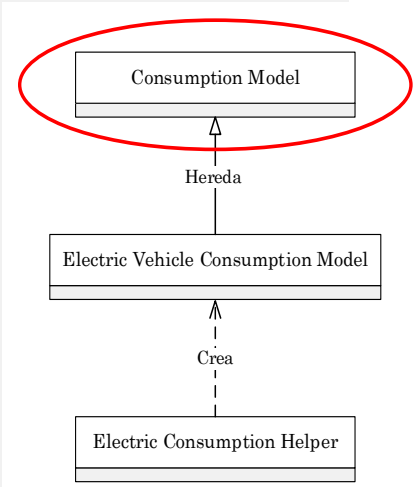
[46] J. Cobb, «Top Six Plug-in Vehicle Adopting Countries – 2015,» Hybrid Cars, 18 Enero 2016. [En línea].
Disponible en:
http://www.hybridcars.com/top-six-plug-in-vehicle-adopting-countries-2015/
[Último acceso: 14 de junio 2018].

[47] E. J. Gago, «Electric vehicles: Case study for Spain,» de *Electric Vehicles: Prospects and Challenges*, 2017, p. 287.

[48] Institute of Transportation Systems at the German Aerospace Center, «SUMO - Simulation of Urban MObility,» SUMO, 2011. [En línea].
Disponible en:
http://sumo.dlr.de/index.html
[Último acceso: 14 de junio 2018].

11 ANEXOS

ConsumptionModel.h



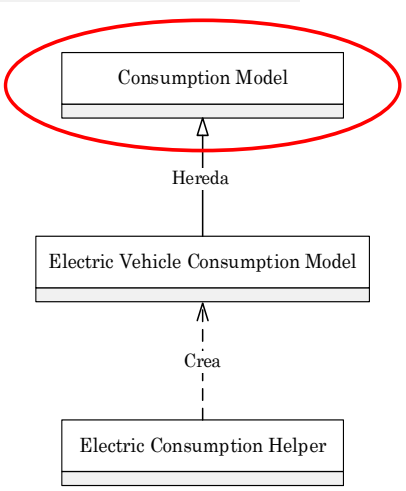

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2018 Unizar
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This 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.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * Author: Samuel Salvatella <ssalvatellaperez@gmail.com>
 */

#ifndef CONSUMPTION_H
#define CONSUMPTION_H

#include "ns3/object.h"
#include "ns3/ptr.h"
#include "ns3/type-id.h"
#include "ns3/mobility-module.h"

namespace ns3
{

/**
 * \defgroup consumption Consumption Models
 *
 */

/**
 * \ingroup consumption
 *
 * \brief Consumption model base class.
 *
 * This is the base class for consumption models. The consumption models perform
 * calculation of the consumption of a vehicle simulated by a mobility. Consumption
 * models update the energy or the fuel by implementing the update function depending
```

```
 * on the type of vehicle (electric, gasoil, gasoline, etc.).
 *
 * The model uses the mobility of the node to perform the calculations keeping
 * the position and speed since the last update.
 *
 */

class ConsumptionModel : public Object
{
public:
    static TypeId GetTypeId (void);
    ConsumptionModel ();
    virtual ~ConsumptionModel ();

    /**
     * This function is called every update interval defined in the corresponding
   . * helper in order to update the consumption of the vehicle.
     */
    virtual void UpdateConsumption (void) = 0;

    /**
     * \brief Sets pointer to node containing this ConsumptionModel.
     *
     * \param node Pointer to node containing this ConsumptionModel.
     */
    void SetNode (Ptr<Node> node);

    /**
     * \returns Pointer to node containing this ConsumptionModel.
     *
     * When a subclass needs to get access to the underlying node base class to
     * print the nodeId for example, it can invoke this method.
     */
    Ptr<Node> GetNode (void) const;

    /**
     * \brief Sets pointer to mobility content in the node.
     *
     * \param pointer to mobility containing in the node.
    */
    void SetMobilityModel (Ptr<const MobilityModel> mobilityModel);

    /**
     * \returns Pointer to mobility content in the node.
     *
     * When a subclass needs to get access to the underlying mobility base class to
     * get information about the mobility.
     */
    Ptr<const MobilityModel> GetMobilityModel (void);
```

```
    /**
     * \brief Sets vector of the last position.
     *
     * \param vector of the last position.
    */
    void SetLastPosition (Vector lastPosition);

    /**
     * \returns Vector of the last position from last update.
     *
     * Get the vector of the last posititon from last update.
     */
    Vector GetLastPosition (void);

    /**
     * \brief Sets vector of the last velocity.
     *
     * \param vector of the last velocity.
    */
    void SetLastVelocity (Vector lastVelocity);

    /**
     * \returns Vector of the last velocity from last update.
     *
     * Get the vector of the last velocity from last update.
     */
    Vector GetLastVelocity (void);

    /**
     * \brief Sets the time of the last update.
     *
     * \param time of the last velocity.
    */
    void SetLastUpdateTime (Time lastUpdate);

    /**
     * \returns Time of the last update.
     *
     * Get the time of the last update.
     */
    Time GetLastUpdateTime (void);

private:

    /**
     * Pointer to node containing this Consumption Model. Used by helper class to make
     * sure models are installed onto the corresponding node.
     */
```

```
    Ptr<Node> m_node;

protected:

    /**
     * Pointer to mobility content in the node. Used for the update
     * for calculations of consumption.
     */
    Ptr<const MobilityModel> m_mobilityModel;

    /**
     * Vector of the last position (x, y, z).
     */
    Vector m_lastPosition;

    /**
     * Vector of the last velocity (x, y, z) expressed in m/s.
     */
    Vector m_lastVelocity;

    /**
     * Time of the last update.
     */
    Time m_lastUpdateTime;

};

}

#endif /* CONSUMPTION_H */
```

ConsumptionModel.cc

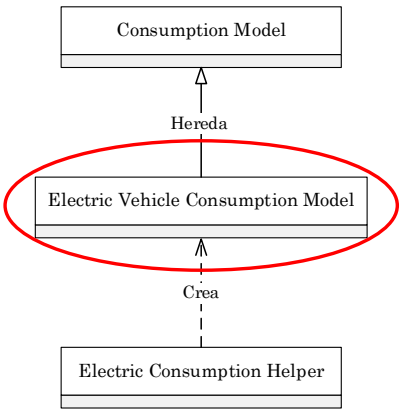



```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2018 Unizar
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This 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.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Samuel Salvatella <ssalvatellaperez@gmail.com>
 */

#include "ns3/core-module.h"
#include "ns3/log.h"
#include "ns3/ptr.h"
#include "ns3/assert.h"
#include "ns3/double.h"
#include "ns3/type-id.h"
#include "ns3/mobility-module.h"
#include "consumption-model.h"

namespace ns3
{

    NS_LOG_COMPONENT_DEFINE ("ConsumptionModel");

    NS_OBJECT_ENSURE_REGISTERED (ConsumptionModel);

    TypeId
    ConsumptionModel::GetTypeId (void)
    {
        static TypeId tid = TypeId ("ns3::ConsumptionModel")
        .SetParent<Object> ()
        .SetGroupName ("Consumption")
        ;
        return tid;
    }

    ConsumptionModel::ConsumptionModel ()
```

```
{
    NS_LOG_FUNCTION (this);
}

ConsumptionModel::~ConsumptionModel ()
{
    NS_LOG_FUNCTION (this);
}

/*
* Private functions start here.
*/

void
ConsumptionModel::SetNode (Ptr<Node> node)
{
    NS_LOG_FUNCTION (this);
    NS_ASSERT (node != NULL);
    m_node = node;
}

Ptr<Node>
ConsumptionModel::GetNode (void) const
{
    return m_node;
}

Ptr<const MobilityModel>
ConsumptionModel::GetMobilityModel (void)
{
    return m_mobilityModel;
}

void
ConsumptionModel::SetMobilityModel (Ptr<const MobilityModel> model)
{
    NS_LOG_FUNCTION (this);
    NS_ASSERT (model != NULL);
    m_mobilityModel = model;
}

void
ConsumptionModel::SetLastPosition (Vector lastPosition)
{
    NS_LOG_FUNCTION (this);
    m_lastPosition = lastPosition;
}

Vector
```

```
    ConsumptionModel::GetLastPosition (void)
    {
        return m_lastPosition;
    }

    void
    ConsumptionModel::SetLastVelocity (Vector lastVelocity)
    {
        NS_LOG_FUNCTION (this);
        m_lastVelocity = lastVelocity;
    }

    Vector
    ConsumptionModel::GetLastVelocity (void)
    {
        return m_lastVelocity;
    }

    void
    ConsumptionModel::SetLastUpdateTime (Time lastUpdate)
    {
        NS_LOG_FUNCTION (this);
        m_lastUpdateTime = lastUpdate;
    }

    Time
    ConsumptionModel::GetLastUpdateTime (void)
    {
        return m_lastUpdateTime;
    }

}
```

ElectricVehicleConsumptionModel.h




```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2018 Unizar
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * 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.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Samuel Salvatella <ssalvatellaperez@gmail.com>
 */

#ifndef ELECTRIC_VEHICLE_CONSUMPTION_MODEL_H
#define ELECTRIC_VEHICLE_CONSUMPTION_MODEL_H

#include "ns3/traced-value.h"
#include "consumption-model.h"
#include "ns3/nstime.h"
#include "ns3/event-id.h"
#include "ns3/mobility-module.h"

#define STANDARD_GRAVITY 9.80665
#define DENSITY_AIR 1.2041
#define JOULES_TO_WH 0.0002778

namespace ns3 {

/**
 * \ingroup consumption
 * \brief Model consumption of electric vehicle based on [1].
 *
 * The values of the model are loaded by the corresponding help class from an XML.
 * The default values are set to zero.
 *
 * Energy consumed each update can be trace from the attribute RemainingEnergyWh.
 *
 * The model requires several parameters for simulate consumption:
 * - InitialEnergyWh, initial energy when start the simulation, in Wh
 * - VehicleMass, mass of the vehicle in Kg
```

```
 * - MaximumBatteryCapacity, maximum capacity of the vehicle battery, in Wh
 * - FrontSurfaceArea, front surface area of the vehicle, in m2
 * - AirDragCoefficient, air drag coefficient of the vehicle
 * - InternalMomentOfInertia, internal moment of intertia of the vehicle, in Kg * m2
 * - RadialDragCoefficient, radial drag coefficient of the vehicle
 * - RollDragCoefficient, roll drag coefficient of the vehicle
 * - ConstantPowerIntake, constant power intake of the vehicle (lights, air conditioner, etc.),
in W
 * - PropulsionEfficiency, propulsion efficiency factor
 * - RecuperationEfficiency, recuperation efficiency factor
 *
 * References:
 * [1] Kurczveil, T., López, P.A., Schnieder, E., Implementation of an Energy Model and a
Charging Infrastructure in SUMO.
 *
 */
  class ElectricVehicleConsumptionModel : public ConsumptionModel
  {
  public:

    static TypeId GetTypeId (void);

    ElectricVehicleConsumptionModel ();

    ~ElectricVehicleConsumptionModel (void);

    /**
     * This function is called every update interval defined in the corresponding
     * helper in order to update the consumption of the vehicle.
    */
    virtual void UpdateConsumption (void);

  private:

    /**
     * \returns Double with the difference of battery energy between a moment [k] and [k + 1]
     *
     * \brief Calculate the diferrence of battery energy.
     */
    double CalculateEnergyDiff (void);

    /**
     * Save the position and velocity of the momento to use in the next update
     * consumption call.
    */
    void SaveLastPosAndVel (void);

    /**
     * \returns Double distance in meters between two position vectors.
```

```
   *
   * \brief Calculate the Euclidean distance between two vectors in meters.
   */
  double GetDistance (Vector u, Vector v);

  /**
   * \param Double velocity in m/s.
   *
   * \returns Double distance in meters
   *
   * \brief Calculate the distance traveled by the vehicle since the last update from a speed.
   */
  double VelocityToDistance (double velocity);

  /**
   * \param Vector of a velocity
   *
   * \returns Double steering angle of the vehicle in radians
   *
   * \brief Calculate the steering angle of the vehicle in radians
   */
  double GetAngle (Vector v);

  /**
   * \returns Double steering angle of the vehicle in radians
   *
   * Calculate the steering angle of the vehicle in radians
   */
  double GetAngleDiff (double angle1, double angle2);

  /*
  * Getters and Setters
  */
public:

  double GetInitialEnergy (void) const;

  void SetInitialEnergy (double initialEnergyWh);

  double GetSupplyVoltage (void) const;

  double GetRemainingEnergy (void);

  void SetRemainingEnergy (double remainingEnergy);

  void DecreaseRemainingEnergy (double energyDecrease);

  void IncreaseRemainingEnergy (double energyIncrease);
```

```
  double GetTotalEnergyConsumed (void);

  void SetTotalEnergyConsumed (double energyConsumed);

  double GetEnergyConsumed (void);

  void SetEnergyConsumed (double energyConsumed);

  void IncreaseTotalEnergyConsumed (double energyConsumed);

  double GetEnergyFraction (void);

  double GetMaximunBatteryCapacity (void);

  void SetMaximunBatteryCapacity (double maximunBatteryCapacity);

  double GetVehicleMass (void);

  void SetVehicleMass (double vehicleMass);

  double GetFrontSurfaceArea (void);

  void SetFrontSurfaceArea (double frontSurfaceArea);

  double GetAirDragCoefficient (void);

  void SetAirDragCoefficient (double airDragCoefficient);

  double GetInternalMomentOfInertia (void);

  void SetInternalMomentOfInertia (double internalMomentOfInertia);

  double GetRadialDragCoefficient (void);

  void SetRadialDragCoefficient (double radialDragCoefficient);

  double GetRollDragCoefficient (void);

  void SetRollDragCoefficient (double rollDragCoefficient);

  double GetConstantPowerIntake (void);

  void SetConstantPowerIntake(double constantPowerIntake);

  double GetPropulsionEfficiency (void);

  void SetPropulsionEfficiency (double propulsionEfficiency);

  double GetRecuperationEfficiency (void);
```

```
    void SetRecuperationEfficiency (double recuperationEfficiency);

    double GetVelocity (void);

    double GetVelocity (Vector vel);

  private:
    double m_initialEnergyWh;                    // initial energy in Wh
    TracedValue<double> m_remainingEnergyWh;     // remaining energy in Wh
    double m_totalEnergyConsumed;                // total energy consumed in Wh
    double m_energyConsumed;                     // energy consumed in last update in Wh
    double m_maximumBatteryCapacity;             // maximum battery capacity in Wh
    double m_vehicleMass;                        // vehicle mass in Kg
    double m_frontSurfaceArea;                   // front surface area of vehicle in m2
    double m_airDragCoefficient;                 // air drag coefficient
    double m_internalMomentOfInertia;            // internal moment of inertia in Kg * m2
    double m_radialDragCoefficient;              // radial drag coefficicent
    double m_rollDragCoefficient;                // roll drag coefficient
    double m_constantPowerIntake;                // constant power intake of vehicle in W
    double m_propulsionEfficiency;               // propulsion efficiency factor
    double m_recuperationEfficiency;             // recuperation efficiency factor
    double m_lastAngle;                          // last angle of vehicle in degrees
    Time m_timeFromLastUpdate;
  };

} // namespace ns3

#endif /* ELECTRIC_VEHICLE_CONSUMPTION_MODEL_H */
```

ElectricVehicleConsumptionModel.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2018 Unizar
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * 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.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Samuel Salvatella <ssalvatellaperez@gmail.com>
 */

#include <cmath>

#include "ns3/log.h"
#include "ns3/assert.h"
#include "ns3/double.h"
#include "ns3/trace-source-accessor.h"
#include "electric-vehicle-consumption-model.h"
#include "ns3/simulator.h"

namespace ns3 {

    NS_LOG_COMPONENT_DEFINE ("ElectricVehicleConsumptionModel");

    NS_OBJECT_ENSURE_REGISTERED (ElectricVehicleConsumptionModel);

    TypeId
    ElectricVehicleConsumptionModel::GetTypeId (void)
    {
      static TypeId tid = TypeId ("ns3::ElectricVehicleConsumptionModel")
        .SetParent<ConsumptionModel> ()
        .SetGroupName ("Consumption")
        .AddConstructor<ElectricVehicleConsumptionModel> ()
        .AddTraceSource ("RemainingEnergy",
                         "Remaining energy in vehicle in Wh.",
                         MakeTraceSourceAccessor
                                     (&ElectricVehicleConsumptionModel::m_remainingEnergyWh),
                                          "ns3::TracedValueCallback::Double")
```

```
  ;
  return tid;
}

ElectricVehicleConsumptionModel::ElectricVehicleConsumptionModel ()
{
  NS_LOG_FUNCTION (this);
}

ElectricVehicleConsumptionModel::~ElectricVehicleConsumptionModel ()
{
  NS_LOG_FUNCTION (this);
}

void
ElectricVehicleConsumptionModel::UpdateConsumption (void)
{

  m_timeFromLastUpdate = Simulator::Now () - GetLastUpdateTime ();

  NS_LOG_FUNCTION (this);
  NS_LOG_DEBUG ("ElectricVehicleConsumptionModel:Updating remaining energy at node #" <<
                GetNode ()->GetId ());

  // do not update if simulation has finished
  if (Simulator::IsFinished ())
  {
    return;
  }

  double energyDiff = CalculateEnergyDiff (); // Wh

  DecreaseRemainingEnergy (energyDiff);
  SetEnergyConsumed (energyDiff);
  IncreaseTotalEnergyConsumed (energyDiff);

  SetLastUpdateTime (Simulator::Now ());
  SaveLastPosAndVel ();
  m_lastAngle = GetAngle (GetMobilityModel ()->GetVelocity ());
}

double ElectricVehicleConsumptionModel::CalculateEnergyDiff (void)
{

  double velocityNow = GetVelocity (m_mobilityModel->GetVelocity ());
  double lastVelocity = GetVelocity (m_lastVelocity);
  double heightNow = m_mobilityModel->GetPosition ().z;
  double lastHeight = m_lastPosition.z;
```

```
    double distanceCovered = VelocityToDistance (velocityNow);

    // calculate potential energy difference
    double energyDiff = GetVehicleMass () * STANDARD_GRAVITY * (heightNow - lastHeight);

    // kinetic energy difference of vehicle
    energyDiff += 0.5 * GetVehicleMass ()
                  * (velocityNow * velocityNow - lastVelocity * lastVelocity);

    // add rotational energy diff of internal rotating elements
    energyDiff += GetInternalMomentOfInertia ()
                  * (velocityNow * velocityNow - lastVelocity * lastVelocity);

    // Energy loss through Air resistance [Ws]
    // Calculate energy losses:
    // EnergyLoss,Air = 1/2 * rho_air [kg/m^3] * myFrontSurfaceArea [m^2] *
     //                          myAirDragCoefficient [-] * v_Veh^2 [m/s] * s [m]
    //                    ... with rho_air [kg/m^3] = 1,2041 kg/m^3 (at T = 20C)
    //                    ... with s [m] = v_Veh [m/s] * TS [s]
    energyDiff += 0.5 * DENSITY_AIR * GetFrontSurfaceArea () * GetAirDragCoefficient () *
                                              velocityNow * velocityNow * distanceCovered;

    // Energy loss through Roll resistance [Ws]
    //                    ... (fabs(veh.getSpeed())>=0.01) = 0, if vehicle isn't moving
    // EnergyLoss,Tire = c_R [-] * F_N [N] * s [m]
    //                    ... with c_R = ~0.012    (car tire on asphalt)
    //                    ... with F_N [N] = myMass [kg] * g [m/s^2]
    energyDiff += GetRollDragCoefficient () * STANDARD_GRAVITY * GetVehicleMass () *
                                                                          distanceCovered;

    // Energy loss through friction by radial force [Ws]
    // If angle of vehicle was changed
    const double angleDiff = m_lastAngle == std::numeric_limits<double>::infinity() ? 0. :
            GetAngleDiff(m_lastAngle, GetAngle (GetMobilityModel ()->GetVelocity ()));

    if (angleDiff != 0.)
    {
      // Compute new radio
      double radius = distanceCovered / fabs(angleDiff);

      // Check if radius is in the interval [0.0001 - 10000]
      // (To avoid overflow and division by zero)
      if (radius < 0.0001)
      {
          radius = 0.0001;
      } else if (radius > 10000)
      {
          radius = 10000;
      }
```

```
    // EnergyLoss,internalFrictionRadialForce = c [m] * F_rad [N];
    // Energy loss through friction by radial force [Ws]
    energyDiff += GetRadialDragCoefficient () * GetVehicleMass ()
                        * velocityNow * velocityNow / radius;
  }

  // EnergyLoss,constantConsumers
  // Energy loss through constant loads (e.g. A/C) [Ws]
  energyDiff += GetConstantPowerIntake ();

  //E_Bat = E_kin_pot + EnergyLoss;
  if (energyDiff > 0)
  {
    energyDiff /= GetPropulsionEfficiency ();
  } else
  {
    energyDiff += GetRecuperationEfficiency ();
  }

  // convert from [Ws] to [Wh] (3600s / 1h):
  return energyDiff / 3600;
}

void
ElectricVehicleConsumptionModel::SaveLastPosAndVel (void)
{
  m_lastPosition.x = m_mobilityModel->GetPosition ().x;
  m_lastPosition.y = m_mobilityModel->GetPosition ().y;
  m_lastPosition.z = m_mobilityModel->GetPosition ().z;

  m_lastVelocity.x = m_mobilityModel->GetVelocity ().x;
  m_lastVelocity.y = m_mobilityModel->GetVelocity ().y;
  m_lastVelocity.z = m_mobilityModel->GetVelocity ().z;
}

double
ElectricVehicleConsumptionModel::VelocityToDistance (double velocity)
{
  return velocity * m_timeFromLastUpdate.GetSeconds ();
}

double
ElectricVehicleConsumptionModel::GetDistance (Vector u, Vector v)
{
  return std::sqrt ((std::pow (u.x - v.x, 2)
   + std::pow (u.y - v.y, 2) + std::pow (u.z - v.z, 2)));
}
```

```
double
ElectricVehicleConsumptionModel::GetAngle (Vector v)
{
  return std::atan2 (v.y, v.x);
}

double
ElectricVehicleConsumptionModel::GetAngleDiff (double angle1, double angle2)
{
  double dtheta = angle2 - angle1;
  while (dtheta > (double) M_PI) {
      dtheta -= (double)(2.0 * M_PI);
  }
  while (dtheta < (double) - M_PI) {
      dtheta += (double)(2.0 * M_PI);
  }
  return dtheta;
}

/*
* Getters and Setters
*/

double
ElectricVehicleConsumptionModel::GetVelocity (void)
{
  Vector vel = m_mobilityModel->GetVelocity ();
  return std::sqrt (std::pow(vel.x, 2) + std::pow(vel.y, 2) + std::pow(vel.z, 2));
}

double
ElectricVehicleConsumptionModel::GetVelocity (Vector vel)
{
  return std::sqrt (std::pow(vel.x, 2) + std::pow(vel.y, 2) + std::pow(vel.z, 2));
}

double
ElectricVehicleConsumptionModel::GetInitialEnergy (void) const
{
  NS_LOG_FUNCTION (this);
  return m_initialEnergyWh;
}
```

```
void
ElectricVehicleConsumptionModel::SetInitialEnergy (double initialEnergyWh)
{
  NS_LOG_FUNCTION (this << initialEnergyWh);
  // check if initial energy isn't bigger than maximum capacity
  NS_ASSERT (initialEnergyWh >= 0);
  m_initialEnergyWh = initialEnergyWh;
  // set remaining energy to be initial energy
  m_remainingEnergyWh = initialEnergyWh;
}

double
ElectricVehicleConsumptionModel::GetSupplyVoltage (void) const
{
  NS_LOG_FUNCTION (this);
  return 0;
}

double
ElectricVehicleConsumptionModel::GetRemainingEnergy (void)
{
  NS_LOG_FUNCTION (this);
  return m_remainingEnergyWh;
}

void
ElectricVehicleConsumptionModel::SetRemainingEnergy (double remainingEnergyWh)
{

  NS_LOG_FUNCTION (this << remainingEnergyWh);
  m_remainingEnergyWh = remainingEnergyWh;
}

void
ElectricVehicleConsumptionModel::DecreaseRemainingEnergy (double energyDecrease)
{
  NS_LOG_FUNCTION (this << energyDecrease);
  SetRemainingEnergy (GetRemainingEnergy() - energyDecrease);
}

void
ElectricVehicleConsumptionModel::IncreaseRemainingEnergy (double energyIncrease)
{
  NS_LOG_FUNCTION (this << energyIncrease);
  SetRemainingEnergy (GetRemainingEnergy() + energyIncrease);
}
```

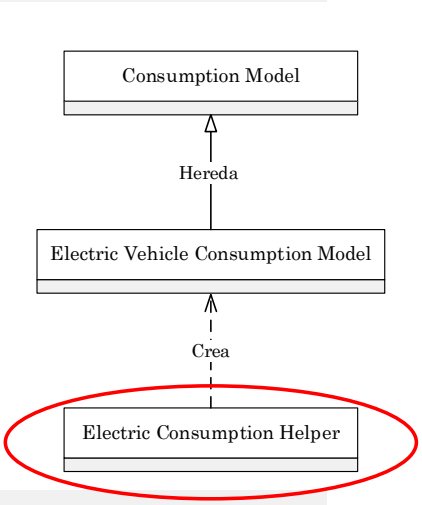
```
double
ElectricVehicleConsumptionModel::GetTotalEnergyConsumed (void)
{
  NS_LOG_FUNCTION (this);
  return m_totalEnergyConsumed;
}


void
ElectricVehicleConsumptionModel::SetTotalEnergyConsumed (double energyConsumed)
{
  NS_LOG_FUNCTION (this << energyConsumed);
  m_totalEnergyConsumed = energyConsumed;
}

double
ElectricVehicleConsumptionModel::GetEnergyConsumed (void)
{
  NS_LOG_FUNCTION (this);
  return m_energyConsumed;
}

void
ElectricVehicleConsumptionModel::SetEnergyConsumed (double energyConsumed)
{
  NS_LOG_FUNCTION (this << energyConsumed);
  m_energyConsumed = energyConsumed;
}

void
ElectricVehicleConsumptionModel::IncreaseTotalEnergyConsumed (double energyConsumed)
{
  NS_LOG_FUNCTION (this << energyConsumed);
  SetTotalEnergyConsumed (GetTotalEnergyConsumed () + energyConsumed);
}

double
ElectricVehicleConsumptionModel::GetEnergyFraction (void)
{
  NS_LOG_FUNCTION (this);
  return GetRemainingEnergy () / GetMaximunBatteryCapacity ();
}

double
ElectricVehicleConsumptionModel::GetMaximunBatteryCapacity (void)
{
  return m_maximumBatteryCapacity;
}
```

```
void
ElectricVehicleConsumptionModel::SetMaximunBatteryCapacity (double maximunBatteryCapacity)
{
  m_maximumBatteryCapacity = maximunBatteryCapacity;
}



double
ElectricVehicleConsumptionModel::GetVehicleMass (void)
{
  return m_vehicleMass;
}

void
ElectricVehicleConsumptionModel::SetVehicleMass (double vehicleMass)
{
  m_vehicleMass = vehicleMass;
}

double
ElectricVehicleConsumptionModel::GetFrontSurfaceArea (void)
{
  return m_frontSurfaceArea;
}

void
ElectricVehicleConsumptionModel::SetFrontSurfaceArea (double frontSurfaceArea)
{
  m_frontSurfaceArea = frontSurfaceArea;
}

double
ElectricVehicleConsumptionModel::GetAirDragCoefficient (void)
{
  return m_airDragCoefficient;
}

void
ElectricVehicleConsumptionModel::SetAirDragCoefficient (double airDragCoefficient)
{
  m_airDragCoefficient = airDragCoefficient;
}

double
ElectricVehicleConsumptionModel::GetInternalMomentOfInertia (void)
{
  return m_internalMomentOfInertia;
}
```

```
void
ElectricVehicleConsumptionModel::SetInternalMomentOfInertia (double internalMomentOfInertia)
{
  m_internalMomentOfInertia = internalMomentOfInertia;
}


double
ElectricVehicleConsumptionModel::GetRadialDragCoefficient (void)
{
  return m_radialDragCoefficient;
}

void
ElectricVehicleConsumptionModel::SetRadialDragCoefficient (double radialDragCoefficient)
{
  m_radialDragCoefficient = radialDragCoefficient;
}

double
ElectricVehicleConsumptionModel::GetRollDragCoefficient (void)
{
  return m_rollDragCoefficient;
}

void
ElectricVehicleConsumptionModel::SetRollDragCoefficient (double rollDragCoefficient)
{
  m_rollDragCoefficient = rollDragCoefficient;
}

double
ElectricVehicleConsumptionModel::GetPropulsionEfficiency (void)
{
  return m_propulsionEfficiency;
}

void
ElectricVehicleConsumptionModel::SetPropulsionEfficiency (double propulsionEfficiency)
{
  m_propulsionEfficiency = propulsionEfficiency;
}

double
ElectricVehicleConsumptionModel::GetRecuperationEfficiency (void)
{
  return m_recuperationEfficiency;
}
```

```
  void
  ElectricVehicleConsumptionModel::SetRecuperationEfficiency (double recuperationEfficiency)
  {
    m_recuperationEfficiency = recuperationEfficiency;
  }


  double
  ElectricVehicleConsumptionModel::GetConstantPowerIntake (void)
  {
    return m_constantPowerIntake;
  }

  void
  ElectricVehicleConsumptionModel::SetConstantPowerIntake(double constantPowerIntake)
  {
    m_constantPowerIntake = constantPowerIntake;
  }

}
```

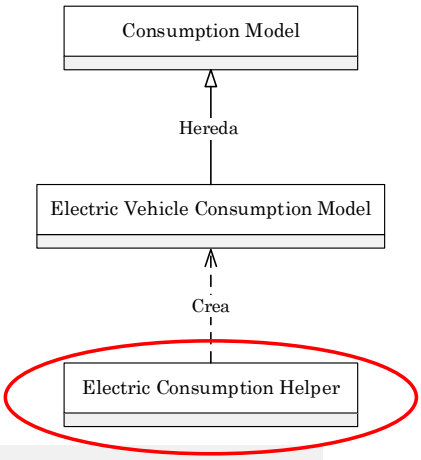
ElectricConsumptionHelper.h




```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2018 Unizar
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * 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.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Samuel Salvatella <ssalvatellaperez@gmail.com>
 */

#ifndef ELECTRIC_CONSUMPTION_HELPER_H
#define ELECTRIC_CONSUMPTION_HELPER_H

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <libxml/encoding.h>
#include <libxml/xmlwriter.h>

#include "electric-vehicle-consumption-model.h"

namespace ns3 {

class ElectricConsumptionHelper
{
public:


  /**
   * \param filename filename of file which contains the
   *        xml with vehicles attributes
   */
  ElectricConsumptionHelper (std::string filename, double updateTime);
```

```
  /**
   * Read the xml file and configure the movement
   * patterns of all nodes contained in the global ns3::NodeList
   * whose nodeId is matches the nodeId of the nodes in the trace
   * file.
   */
  void Install (void);


private:
  void LoadXml (void);
  void CreateModelFromXml (xmlNode * xmlNode);

private:
  std::string m_filename;  // filename of file containing the vehicle attributes
  xmlDoc *m_xmlDoc;        // XML Document with XML nodes
  double m_updateTime;     // time between each update of electric vehicle consumption
};

  Ptr<Node> GetNodeFromContext(std::string context);
  std::string Convert(const xmlChar * xmlChar);


} // namespace ns3

#endif /* ELECTRIC_CONSUMPTION_HELPER_H */
```

ElectricConsumptionHelper.cc




```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2018 Unizar
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * 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.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Samuel Salvatella <ssalvatellaperez@gmail.com>
 */

#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <libxml/encoding.h>
#include <libxml/xmlwriter.h>

#include "ns3/core-module.h"
#include "ns3/node-list.h"
#include "ns3/node.h"
#include "ns3/log.h"
#include "ns3/mobility-module.h"
#include "electric-consumption-helper.h"

namespace ns3 {


  NS_LOG_COMPONENT_DEFINE ("ElectricConsumptionHelper");

  ElectricConsumptionHelper::ElectricConsumptionHelper (std::string filename, double updateTime)
    : m_filename (filename)
  {
    std::ifstream file (m_filename.c_str (), std::ios::in);
    if (updateTime <= 0.01 || std::isnan (updateTime))
    {
      m_updateTime = 1;
    } else
```

```
    {
      m_updateTime = updateTime;
    }

    if (!(file.is_open ())) NS_FATAL_ERROR("Could not open vehicule attributes file " <<
                              m_filename.c_str() << " for reading, aborting here \n");
  }

  void
  ElectricConsumptionHelper::Install (void)
  {
    LoadXml ();
  }

  static void
  UpdateModelConsumption (Ptr<ElectricVehicleConsumptionModel> consumptionModel,
                              double updateTime)
  {
    consumptionModel->UpdateConsumption ();
    Simulator::Schedule (Seconds (updateTime),
       &UpdateModelConsumption, consumptionModel, updateTime);
  }

/*
 * Private functions start here.
 */
  void
  ElectricConsumptionHelper::LoadXml (void)
  {
    m_xmlDoc = NULL;
    /*parse the file and get the DOM */
    m_xmlDoc = xmlReadFile(m_filename.c_str (), NULL, 0);

    // case of failure
    if (m_xmlDoc == NULL)
    {
      NS_LOG_ERROR ("Could not parse XML file " << m_filename);
      return;
    }

    xmlNode *rootElement = xmlDocGetRootElement (m_xmlDoc);
    xmlNode *firstVehicle = rootElement->children;
    xmlNode *currentNode = NULL;

    // create a consumption model for each vehicle especified in XML
    for (currentNode = firstVehicle; currentNode; currentNode = currentNode->next) {
        if (currentNode->type == XML_ELEMENT_NODE) {
            CreateModelFromXml (currentNode);
        }
```

```
  }

  /*free the document */
  xmlFreeDoc(m_xmlDoc);

  /*
   *Free the global variables that may
   *have been allocated by the parser.
   */
  xmlCleanupParser();

}

void
ElectricConsumptionHelper::CreateModelFromXml (xmlNode * xmlVehicleNode)
{
  xmlNode *currentNode = NULL;

  Ptr<ElectricVehicleConsumptionModel> consumptionModel =
     CreateObject<ElectricVehicleConsumptionModel> ();
  uint32_t nodeId;

  // read the node id of vehicle
  for(xmlAttrPtr attr = xmlVehicleNode->properties; NULL != attr; attr = attr->next)
  {
    std::string name = Convert (attr->name);
    if (name == "node")
    {
      nodeId = std::stoi(Convert (attr->children->content));
    } else {
      NS_LOG_ERROR ("Node ID of vehicle is not defined.");
      return;
    }
  }

  Ptr<Node> node = NodeList::GetNode (nodeId);
  if (node == NULL) {
    NS_LOG_ERROR ("Node " << nodeId << " is not created in the simulation.");
    return;
  }

  // read parameters of electric vehicle
  for (currentNode = xmlVehicleNode->children; currentNode; currentNode = currentNode->next) {
    if (currentNode->type == XML_ELEMENT_NODE) {
      std::string clave;
      std::string valor;
      for(xmlAttrPtr attr = currentNode->properties; NULL != attr; attr = attr->next)
      {
        std::string name = Convert (attr->name);
```

```
      if (name == "key")
      {
        clave = Convert (attr->children->content);
      } else if (name == "value")
      {
        valor = Convert (attr->children->content);
      } else
      {
        NS_LOG_ERROR ("Could not parse XML file. Check the XML structure.");
        return;
      }

    }
    if (clave.empty () || valor.empty ())
    {
      NS_LOG_ERROR ("Could not parse XML file. Check the XML structure.");
      return;
    }

    // std::cout << clave << "=" << valor << "\n";

    if (clave == "maximumBatteryCapacity")
    {
      consumptionModel->SetMaximunBatteryCapacity (std::stod (valor));
    } else if (clave == "vehicleMass")
    {
      consumptionModel->SetVehicleMass (std::stod (valor));
    } else if (clave == "frontSurfaceArea")
    {
      consumptionModel->SetFrontSurfaceArea (std::stod (valor));
    } else if (clave == "airDragCoefficient")
    {
      consumptionModel->SetAirDragCoefficient (std::stod (valor));
    } else if (clave == "internalMomentOfInertia")
    {
      consumptionModel->SetInternalMomentOfInertia (std::stod (valor));
    } else if (clave == "radialDragCoefficient")
    {
      consumptionModel->SetRadialDragCoefficient (std::stod (valor));
    } else if (clave == "rollDragCoefficient")
    {
      consumptionModel->SetRollDragCoefficient (std::stod (valor));
    } else if (clave == "constantPowerIntake")
    {
      consumptionModel->SetConstantPowerIntake (std::stod (valor));
    } else if (clave == "propulsionEfficiency")
    {
      consumptionModel->SetPropulsionEfficiency (std::stod (valor));
    } else if (clave == "recuperationEfficiency")
```

```
      {
        consumptionModel->SetRecuperationEfficiency (std::stod (valor));
      } else if (clave == "initialEnergy")
      {
        consumptionModel->SetInitialEnergy (std::stod (valor));
      } else
      {
        NS_LOG_ERROR ("Could not parse XML file. Key " << clave << " unrecognized.");
      }
    }
  }


  //all consumption model has a mobility model
  Ptr<MobilityModel> mobility = node->GetObject<MobilityModel> ();
  consumptionModel->SetMobilityModel (mobility);

  //we add the consumption model to the node
  node->AggregateObject (consumptionModel);

  consumptionModel->UpdateConsumption (); // first update in second 0

  Simulator::Schedule (Seconds (m_updateTime),
     &UpdateModelConsumption, consumptionModel, m_updateTime);
}


Ptr<Node>
GetNodeFromContext (std::string context)
{
  std::string delimiter = "/";

  size_t i = 0;
  size_t count = 0;
  std::string token;
  while ((i = context.find(delimiter)) != std::string::npos) {
      count++;
      token = context.substr(0, i);
      context.erase(0, i + delimiter.length());
      if (count == 3)
      {
        return NodeList::GetNode (std::stoi(token));
      }
  }
  return NULL;
}
```

```
  std::string
  Convert(const xmlChar * xmlChar)
  {
    char* a = (char *)xmlChar;
    return std::string(a);
  }

} // namespace ns3
```

ElectricConsumption.cc

```
/* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2018
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * 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.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Samuel Salvatella <ssalvatellaperez@gmail.com>
 *
 *
 * This example demonstrates the use of ElectricMobilityHelper class to work with mobility.
 *
 * Detailed example description.
 *
 *  - intended usage: this should be used in order to load ns2 movement trace files and a xml
 *     file that specifies the attributes of each electric vehicle to simulate the consumption of
 *     each vehicle associated with a node of the traces
 *  - behavior:
 *      - ElectricMobilityHelper object is created, associated to the specified trace file.
 *      - A log file is created, using the log file name argument.
 *      - A node container is created with the number of nodes specified in the command line.
For the default ns-2 trace, specify the value 2 for this argument.
 *      - the program calls the Install() method of ElectricMobilityHelper to set mobility to
 *        nodes. At this moment, the file is read line by line, and the movement is scheduled in
 *        the simulator.
 *      - A callback is configured, so each time a node changes its course a log message is
printed.
 *  - expected output: example prints out messages generated by each read line from the ns2
 *                     movement trace file.
 *  For each line, it shows if the line is correct, or of it has errors and in this case it will
 *  be ignored.
 *
 * Usage of electric-mobility:
 *
 *  ./waf --run "electric-mobility \
 *        --traceFile=src/mobility/examples/default.ns_movements --
vehicleAttributes=src/mobility/examples/vehicleAttributes.xml
```

```
 *       --nodeNum=2  --duration=100.0 --logFile=electric-mobility.log"
 *
 *  NOTE: ns2-traces-file could be an absolute or relative path. You could use the file
 *       default.ns_movements
 *       included in the same directory as the example file.
 *  NOTE 2: Number of nodes present in the trace file must match with the command line argument.
 *         Note that you must know it before to be able to load it.
 *  NOTE 3: Duration must be a positive number and should match the trace file. Note that you
 *         must know it before to be able to load it.
 */

#include <iostream>
#include <fstream>
#include <sstream>

#include "ns3/log.h"
#include "ns3/node-list.h"
#include "ns3/node.h"
#include "ns3/core-module.h"
#include "ns3/mobility-module.h"
#include "ns3/mobility-module.h"
#include "ns3/ns2-mobility-helper.h"
#include "electric-consumption-helper.h"

using namespace ns3;



void RemainingEnergyTrace (std::string context, double previousEnergy, double currentEnergy)
{
  Ptr<Node> node = GetNodeFromContext(context);
  Ptr<ElectricVehicleConsumptionModel> consumptionModel = node-
>GetObject<ElectricVehicleConsumptionModel> ();
  Ptr<const MobilityModel> mobilityModel = consumptionModel->GetMobilityModel ();
  Vector pos = mobilityModel->GetPosition ();

  NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t"
    << node->GetId () << "\t"
    << pos.x << "\t"
    << pos.y << "\t"
    << pos.z << "\t"
    << consumptionModel->GetVelocity () << "\t"
    << consumptionModel->GetEnergyFraction () << "\t"
    << currentEnergy << "\t"
    << consumptionModel->GetEnergyConsumed () << "\t"
    << consumptionModel->GetTotalEnergyConsumed ());
}
```

```
// Example to use ns2 traces file and xml file to simulate consumption of electric vehicles
int main (int argc, char *argv[])
{
  std::string traceFile;
  std::string vehicleAttributesFile;

  int    nodeNum;
  double duration;
  double updateTime;

  // Parse command line attribute
  CommandLine cmd;
  cmd.AddValue ("traceFile", "Ns2 movement trace file", traceFile);
  cmd.AddValue ("vehicleAttributes", "Vehicle Attributes", vehicleAttributesFile);
  cmd.AddValue ("nodeNum", "Number of nodes", nodeNum);
  cmd.AddValue ("duration", "Duration of Simulation", duration);
  cmd.AddValue ("updateTime", "Time between each update of electric vehicle consumption.",
updateTime);
  cmd.Parse (argc,argv);

  // Check command line arguments
  if (traceFile.empty () || vehicleAttributesFile.empty () || nodeNum <= 0 || duration <= 0)
    {
      std::cout << "Usage of " << argv[0] << " :\n\n"
      "./waf --run \"electric-mobility"
      " --traceFile=src/mobility/examples/default.ns_movements"
      " --vehicleAttributes=src/mobility/examples/vehicleAttributes.xml"
      " --nodeNum=2 --duration=100.0\" \n\n"
      "NOTE: ns2-traces-file could be an absolute or relative path. You could use the file
default.ns_movements\n"
      "      included in the same directory of this example file.\n\n"
      "NOTE 2: Number of nodes present in the trace file must match with the command line
argument and must\n"
      "        be a positive number. Note that you must know it before to be able to load
it.\n\n"
      "NOTE 3: Duration must be a positive number. Note that you must know it before to be able
to load it.\n\n";

      return 0;
    }

  // Create Ns2MobilityHelper with the specified trace log file as parameter
  Ns2MobilityHelper ns2 = Ns2MobilityHelper (traceFile);

  // Create ElectricConsumptionHelper with the xml of vehicle attributes
  ElectricConsumptionHelper electricMobility = ElectricConsumptionHelper (vehicleAttributesFile,
                                                                          updateTime);
```

```
  // Create all nodes.
  NodeContainer stas;
  stas.Create (nodeNum);

  ns2.Install (); // configure movements for each node, while reading trace file
  electricMobility.Install (); // configure the vehicle attributes for each node

  Config::Connect ("/NodeList/*/$ns3::ElectricVehicleConsumptionModel/RemainingEnergy",
              MakeCallback (&RemainingEnergyTrace));

  // Log a header for data
  NS_LOG_UNCOND("Time \t#\tx\ty\tz\tVel(m/s)\tEnergy Level(%)\tCurrent Energy(Wh)\tEnergy
Consumed(Wh)\tTotal Consumed(Wh)");

  Simulator::Stop (Seconds (duration));
  Simulator::Run ();

  // show final statics
  int i = 0;
  for (i = 0; i < nodeNum; i++)
  {
    Ptr<Node> n = NodeList::GetNode (i);
    Ptr<ElectricVehicleConsumptionModel> model = n->GetObject<ElectricVehicleConsumptionModel>();
    NS_LOG_UNCOND ("Node " << i
       << " Total consumed: " << model->GetTotalEnergyConsumed () << " Wh");
  }

  Simulator::Destroy ();

  return 0;
}
```