Facultad de
Ciencias Sociales y Tecnologías de la Inf.
Talavera de la Reina. UCLM

UNIVERSIDAD DE CASTILLA-LA MANCHA

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo Fin de Grado

DISEÑO E IMPLEMENTACIÓN DE UN GEMELO DIGITAL PARA LA GESTIÓN Y CONTROL DE TRÁFICO URBANO

Design and implementation of a digital twin for urban traffic management and control

Ismael López Marín

Junio de 2026

DISEÑO E IMPLEMENTACIÓN DE UN GEMELO DIGITAL PARA LA GESTIÓN Y CONTROL DE TRÁFICO URBANO

Facultad de
Ciencias Sociales y Tecnologías de la Inf.
Talavera de la Reina. UCLM

UNIVERSIDAD DE CASTILLA-LA MANCHA
Tecnologías y Sistemas de Información

GRADO EN INGENIERÍA INFORMÁTICA

SISTEMAS DE INFORMACIÓN

Trabajo Fin de Grado

DISEÑO E IMPLEMENTACIÓN DE UN GEMELO DIGITAL PARA LA GESTIÓN Y CONTROL DE TRÁFICO URBANO

Design and implementation of a digital twin for urban traffic management and control

Autoría: Ismael López Marín
Tutoría: Luis Cabañero Gómez

Junio de 2026

Ismael López Marín

Talavera de la Reina – España

DECLARACIÓN DE AUTORÍA

D. **Ismael López Marín**, con DNI 50636404H, estudiante del Grado en Ingeniería Informática (con mención en Sistemas de Información) de la Universidad de Castilla-La Mancha,

DECLARA

que el presente Trabajo Fin de Grado, titulado **«DISEÑO E IMPLEMENTACIÓN DE UN GEMELO DIGITAL PARA LA GESTIÓN Y CONTROL DE TRÁFICO URBANO»**, ha sido realizado de forma autónoma y constituye un trabajo original e inédito, fruto de su esfuerzo personal y elaborado bajo la dirección de Luis Cabañero Gómez. Asimismo, declara que se han citado debidamente todas las fuentes consultadas y que el trabajo respeta los derechos de propiedad intelectual de terceros, no incurriendo en plagio ni en copia de obras ajenas. Igualmente, declara que la memoria no ha sido presentada previamente, en su totalidad o en parte, para la obtención de ningún otro título.

Y para que así conste, firma la presente declaración.

Talavera de la Reina, Junio de 2026

LOPEZ MARIN ISMAEL - 50636404H	Firmado por: LOPEZ MARIN ISMAEL - 50636404H Fecha: 2026-06-12 17:43:02 CEST

Fdo.: Ismael López Marín

TRIBUNAL:

Presidencia:

Vocal:

Secretaría:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENCIA	VOCAL	SECRETARÍA
Fdo.:	Fdo.:	Fdo.:

Resumen

Este Trabajo Fin de Grado (TFG) presenta el diseño y la implementación de un *gemelo digital* del tráfico urbano de Talavera de la Reina: una herramienta que reconstruye la red vial real de la ciudad, simula sobre ella el tráfico vehículo a vehículo y permite ensayar escenarios hipotéticos —la entrada en vigor de una zona de bajas emisiones, el corte de una calle por una inundación— observando sus consecuencias en una visualización 3D interactiva.

El sistema se compone de dos piezas que se comunican en tiempo real. Un *backend* en *Python* gobierna la simulación: importa la red desde OpenStreetMap, la almacena en una base de datos geoespacial (PostgreSQL/PostGIS), calcula las rutas con A* y mueve cada vehículo con los modelos microscópicos Intelligent Driver Model (IDM) y Minimizing Overall Braking Induced by Lane changes (MOBIL); sobre ese tráfico base, el operador activa semáforos, incidentes y zonas de bajas emisiones. Un cliente 3D construido sobre *Godot* recibe el estado de la simulación a $3\,\mathrm{Hz}$ —los comandos viajan por una Application Programming Interface (API) Representational State Transfer (REST) y el flujo continuo de estado, por un protocolo WebSocket (WS) binario— y lo presenta a $60\,\mathrm{FPS}$ mediante interpolación y renderizado por instancias.

Sobre esta base se construyen dos casos de estudio tomados de la realidad reciente de Talavera: la Zona de Bajas Emisiones (ZBE) aprobada en 2025-2026, con sus distintos modos de aplicación, y el corte de viales provocado por la inundación del Tajo de febrero de 2026. En cuanto al rendimiento, el sistema sostiene $4\,000$ vehículos simultáneos a $3\,\mathrm{Hz}$ con amplio margen de cómputo y el cliente mantiene los $60\,\mathrm{FPS}$ estables.

Como contribuciones principales, el trabajo entrega un simulador de código abierto reproducible mediante *Docker Compose*, un formato binario de transmisión que reduce a la cuarta parte el ancho de banda frente a JavaScript Object Notation (JSON), una parametrización razonada de IDM/MOBIL para tres tipos de vehículo en entorno urbano español y los dos casos de estudio citados.

Abstract

This *Final Degree Project* (TFG) presents the design and implementation of a *digital twin* of the urban traffic of Talavera de la Reina: a tool that rebuilds the city's real road network, simulates traffic on it vehicle by vehicle, and lets the user try out hypothetical scenarios —a low-emission zone coming into force, a street closed by a flood— while watching their consequences in an interactive 3D visualisation.

The system consists of two pieces communicating in real time. A *Python* backend governs the simulation: it imports the network from OpenStreetMap, stores it in a geospatial database (PostgreSQL/PostGIS), computes routes with A* and moves each vehicle with the microscopic IDM and MOBIL models; on top of that base traffic, the operator activates traffic lights, incidents and low-emission zones. A 3D client built on *Godot* receives the simulation state at $3\,\text{Hz}$ —commands travel over a REST API, and the continuous state stream over a binary WS protocol— and presents it at $60\,\text{FPS}$ through interpolation and instanced rendering.

On this foundation, two case studies taken from Talavera's recent reality are built: the low-emission zone (ZBE) approved in 2025-2026, with its different enforcement modes, and the road closures caused by the Tagus river flood of February 2026. Performance-wise, the system sustains $4\,000$ simultaneous vehicles at $3\,\text{Hz}$ with ample compute headroom, while the client holds a steady $60\,\text{FPS}$.

As its main contributions, the project delivers an open-source simulator reproducible via *Docker Compose*, a binary wire format that cuts bandwidth to a quarter of its JSON equivalent, a reasoned IDM/MOBIL parameterisation for three vehicle types in a Spanish urban setting, and the two case studies above.

Agradecimientos

A quienes me han acompañado durante el Grado en Ingeniería Informática y en particular, durante los meses dedicados a este TFG: gracias.

A mi tutor, Luis, por su disponibilidad, orientación y paciencia para revisar las entregas del gemelo digital, escuchar mis dudas y ayudarme a desbloquear el proyecto cuando más hizo falta. A los profesores de la Facultad de Ciencias Sociales y Tecnologías de la Información de Talavera de la Reina, por la formación en los fundamentos sobre los que se apoya este trabajo.

A mis amigos y compañeros —especialmente a Pablo, Daniel, Marco y Jorge— , por las interminables discusiones técnicas, los retos compartidos y el ánimo en las semanas que parecían no tener fin.

A mi familia —con un pensamiento muy especial para mi abuela— por ser siempre mi ancla y mi mayor motivación.

A Julia, por ser un refugio seguro, por comprender mis ausencias frente a la pantalla y por seguir preguntando por el avance de este proyecto sabiendo que la respuesta iba a ser larga. Gracias por no dejar de preguntar.

Ismael López Marín

Índice general

Índice de cuadros

Índice de figuras

Listado de acrónimos

API	Application Programming Interface
BD	Base de Datos
CECOPAL	Centro de Coordinación Operativa Municipal
CI	Continuous Integration
CORS	Cross-Origin Resource Sharing
CPU	Central Processing Unit
CTM	Cell Transmission Model
DDL	Data Definition Language
DGT	Dirección General de Tráfico
EPSG	European Petroleum Survey Group
ETL	Extract, Transform, Load
FPS	Frames Per Second
FSM	Finite State Machine
GIL	Global Interpreter Lock
GIS	Geographic Information System
GiST	Generalized Search Tree
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUT	Godot Unit Test
HTTP	Hypertext Transfer Protocol
HUD	Heads-Up Display
IDM	Intelligent Driver Model
IPC	Inter-Process Communication
JSON	JavaScript Object Notation
LOD	Level Of Detail

LWR	Lighthill-Whitham-Richards
MITMA	Ministerio de Transportes, Movilidad y Agenda Urbana
MOBIL	Minimizing Overall Braking Induced by Lane changes
OGC	Open Geospatial Consortium
ORM	Object-Relational Mapping
OSM	OpenStreetMap
PEP	Python Enhancement Proposal
PMUS	Plan de Movilidad Urbana Sostenible
RAM	Random Access Memory
RDP	Ramer-Douglas-Peucker
REST	Representational State Transfer
SQL	Structured Query Language
SRID	Spatial Reference System Identifier
SSE	Server-Sent Events
SUMO	Simulation of Urban MObility
TCP	Transmission Control Protocol
TFG	Trabajo Fin de Grado
TraCI	Traffic Control Interface
TTC	Time-To-Collision
UML	Unified Modeling Language
URL	Uniform Resource Locator
UTM	Universal Transverse Mercator
VISSIM	Verkehr In Städten – SIMulationsmodell
WGS84	World Geodetic System 1984
WS	WebSocket
XML	Extensible Markup Language
ZBE	Zona de Bajas Emisiones

Capítulo 1

Introducción

En febrero de 2026, la crecida del Tajo obligó a cortar al tráfico varias calles del centro de Talavera de la Reina [Caz26, Cas26]. ¿Hacia qué calles se desvió el tráfico expulsado del centro? ¿Cuáles absorbieron el rebote sin colapsar? El ayuntamiento no disponía de ninguna herramienta con la que anticipar la respuesta y hubo de gestionar el episodio de forma reactiva. El presente TFG construye esa herramienta universal de simulación de tráfico urbano. A partir de datos cartográficos abiertos, el sistema es capaz de generar automáticamente un *gemelo digital* para la red viaria de cualquier municipio. Para validar su utilidad, este trabajo toma a Talavera de la Reina como caso de estudio práctico.

1.1 Contexto y motivación

El caso de la inundación no es la única anomalía. Entre 2024 y 2026, Talavera atraviesa una etapa inusualmente intensa para su movilidad: la aprobación —y posterior aplazamiento— de una Zona de Bajas Emisiones (ZBE) [Ayu25, CLM26], la humanización de la travesía de la N-502A —siete millones de euros de inversión que incluyen la reorganización de la Avenida de Portugal [Min24b]— y las propias crecidas del Tajo. Cualquiera de estas actuaciones redibuja los flujos de tráfico de la ciudad, y todas comparten un mismo problema de fondo: el ayuntamiento debe decidir sin una forma rápida de anticipar las consecuencias.

Es una tensión habitual en las ciudades medianas españolas, obligadas a aplicar políticas de movilidad alineadas con la normativa europea y estatal pero sin el presupuesto ni la plantilla técnica para evaluar cada medida. Talavera lo ilustra bien: ha invertido cerca de $900\,000\,€$ en la infraestructura de su ZBE —cámaras, paneles y estaciones de calidad del aire— [Cov24] pero no cuenta con un simulador con el que comprobar, por ejemplo, a qué calles se desviaría el tráfico si se excluyera a los camiones del casco urbano. Esa es la necesidad que motiva este trabajo.

1.2 Un gemelo digital para el tráfico urbano

Un *gemelo digital* es una réplica virtual de un sistema físico que evoluciona en paralelo a él [Gri14]. El concepto, nacido en la industria manufacturera y aeroespacial [TCQ+18, GS12], se ha trasladado en los últimos años a la escala de la ciudad [Bat18]. El que aquí se construye es

de tipo *descriptivo*: reconstruye la red vial real de Talavera —importada de datos cartográficos abiertos— y simula sobre ella el tráfico, pero no se realimenta todavía con datos en tiempo real. Es el operador quien introduce los escenarios que desea estudiar.

La herramienta entregada es un simulador que reproduce el tráfico vehículo a vehículo, lo muestra en una escena 3D de manipulación directa —pensada para un operador no técnico— y permite activar en segundos situaciones hipotéticas: la entrada en vigor de la ZBE, el corte de una calle por una inundación o un accidente en una vía principal. Su valor no reside en competir con los grandes simuladores de tráfico profesionales, sino en integrar toda la cadena —del mapa a la simulación y de la simulación a la pantalla— en una sola pieza manejable.

1.3 Estructura del documento

El resto de la memoria se organiza como sigue:

Capítulo 2: Objetivos
: Objetivo general y objetivos parciales del TFG.

Capítulo 3: Antecedentes y estado del arte
: Estado del arte en simulación de tráfico, datos cartográficos, movilidad sostenible, y el hueco que aborda el TFG.

Capítulo 4: Metodología
: Proceso de desarrollo, planificación temporal y herramientas empleadas.

Capítulo 5: Resultados
: Diseño e implementación del sistema, iteración a iteración, hasta el sistema completo en ejecución.

Capítulo 6: Pruebas y validación
: Pruebas, rendimiento y dos casos de estudio sobre Talavera.

Capítulo 7: Conclusiones y trabajo futuro
: Conclusiones, contribuciones, limitaciones y trabajo futuro.

Capítulo 2

Objetivos

2.1 Objetivo general

El objetivo general de este TFG es:

> Desarrollar un gemelo digital modular para la simulación microscópica, visualización interactiva 3D y control en tiempo real del tráfico urbano de Talavera de la Reina, que permita evaluar escenarios *what-if* y comparar políticas de gestión de tráfico.

2.2 Objetivos específicos

A partir del objetivo general se han fijado seis objetivos específicos, que estructuran el resto de la memoria:

1. **Diseñar la arquitectura del sistema** definiendo los componentes principales e interfaces de comunicación, para establecer una base sólida que favorezca la escalabilidad, integración y modularidad del gemelo digital.
2. **Implementar el esquema de base de datos geoespacial**, incluyendo las herramientas necesarias para operar cómodamente con ella, con el fin de almacenar y procesar eficientemente la topología de la infraestructura urbana que sustenta la simulación.
3. **Desarrollar el modelo de comportamiento vehicular, gestor de vehículos y sistema de cálculo de rutas**, para propiciar una simulación microscópica realista y precisa de las dinámicas de tráfico de la ciudad.
4. **Implementar el Sistema de Control y Gestión de Eventos**, desarrollando un gestor de semáforos, gestor de incidentes, gestor de zonas esenciales y apertura y cierre dinámico de carriles, dotando al sistema de la capacidad de alterar dinámicamente el entorno y evaluar escenarios *what-if*.
5. **Implementar la API REST** para el control de la simulación, la infraestructura y la consulta de métricas, con transmisión del estado del tráfico en tiempo real. **Desarrollar un motor de analíticas** que calcule en tiempo real las métricas clave del tráfico (tiempos de viaje, congestión e impacto de incidentes) para monitorizar y evaluar las políticas de gestión.

6. **Desarrollar el módulo de renderizado interactivo de mapas**, que incluya un sistema de cámaras, panel de control y visualización de eventos, para habilitar al usuario no técnico poder hacer uso de la herramienta de forma intuitiva.

El cumplimiento de los seis objetivos se evalúa explícitamente en el Capítulo 7.

2.3 Requisitos del sistema

Los objetivos anteriores se concretan en el conjunto de requisitos que se resume a continuación y que guía el diseño del sistema (Capítulo 5). Se distinguen los requisitos *funcionales* (qué debe hacer el sistema, Cuadro 2.1), los *no funcionales* (atributos de calidad que debe satisfacer, Cuadro 2.2) y los *condicionantes de alcance* (Cuadro 2.3).

ID	Requisito funcional
RF-01	Importar la red vial desde un fichero OSM (proceso ETL), persistirla en la base de datos geoespacial y construir el grafo dirigido en memoria, con independencia del municipio.
RF-02	Simular el tráfico de forma microscópica, vehículo a vehículo, con seguimiento longitudinal y cambios de carril, y con tipos de vehículo de parámetros físicos diferenciados (turismo, motocicleta y camión).
RF-03	Calcular la ruta de cada vehículo mediante A* sobre el grafo, respetando las penalizaciones inducidas por incidentes y zonas.
RF-04	Gobernar el ciclo de vida de la simulación (arranque, pausa, reanudación y parada) y permitir ajustar su configuración en caliente.
RF-05	Controlar la señalización: semáforos de ciclo fijo con anulación manual, y prioridades en cruces no señalizados, *stop*, ceda el paso y rotondas.
RF-06	Gestionar incidentes que cierren dinámicamente aristas o carriles y reenrutar los vehículos afectados.
RF-07	Gestionar zonas restringidas (por ejemplo, una ZBE) con políticas de aplicación diferenciadas por tipo de vehículo.
RF-08	Exponer una API REST para los comandos del operador y difundir el estado del tráfico en tiempo real mediante WS.
RF-09	Calcular las métricas de tráfico clave: tiempos de viaje, nivel de congestión e impacto de incidentes.
RF-10	Visualizar la simulación en 3D interactiva (red, vehículos, incidentes y zonas), con cámara, panel de control (HUD) y modos de operador para componer escenarios *what-if*.

Cuadro 2.1: Requisitos funcionales del sistema

ID	Requisito no funcional
RNF-01	**Rendimiento.** Sostener la frecuencia de *tick* objetivo (3 Hz) con un dimensionamiento de 4 000 a 6 000 vehículos simultáneos.
RNF-02	**Tiempo real.** Renderizar el cliente a 60 FPS, ocultando mediante interpolación la cadencia de actualización del servidor.
RNF-03	**Modularidad.** Mantener separadas la simulación (servidor) y la presentación (cliente), con el backend organizado en capas de dependencias unidireccionales.
RNF-04	**Portabilidad.** Desplegar el backend y la base de datos de forma reproducible mediante contenedores (Docker Compose).
RNF-05	**Observabilidad.** Instrumentar el rendimiento del motor y exponer sus métricas (formato Prometheus y JSON).

Cuadro 2.2: Requisitos no funcionales del sistema

ID	Condicionante
RES-01	Gemelo digital *descriptivo* en el sentido de la § 1.2: sin realimentación con datos del mundo físico en tiempo real.
RES-02	Tráfico sintético, sin aforos reales; el modelo no se calibra con datos de tráfico (los factores de coste de las rutas son constantes de diseño).
RES-03	La API y el canal WS se exponen sin autenticación ni control de acceso, al concebirse el sistema para ejecución local o en una red de confianza.
RES-04	La restricción de zonas se discrimina por tipo de vehículo, no por la etiqueta ambiental de la DGT.
RES-05	Prototipo de código abierto desarrollado por una sola persona en un plazo académico acotado.

Cuadro 2.3: Condicionantes del alcance

2.4 Análisis de requisitos: elicitación, priorización y trazabilidad

Una vez enunciados los requisitos, esta sección documenta el proceso de ingeniería que los sustenta. En primer lugar se describe cómo se obtuvieron y a qué actor responden (§ 2.4.1); a continuación, cómo se priorizaron y bajo qué criterios se consideran satisfechos (§ 2.4.2). Por último, se establece la trazabilidad que vincula cada requisito con su diseño y su validación (§ 2.4.3).

2.4.1 Elicitación y actor del sistema

Al tratarse de un TFG de un único desarrollador, la elicitación de requisitos no siguió un proceso formal de entrevistas con un cliente corporativo, sino que combinó tres fuentes: la

supervisión del tutor académico, quien proporcionó orientación metodológica y ayudó a definir y acotar el alcance del proyecto para garantizar su viabilidad y rigor; el **análisis del dominio normativo** (Capítulo 3), pues la regulación de circulación y de las ZBE determina buena parte de los requisitos funcionales; y el **prototipado iterativo** (§ 4.1), con el que varios requisitos no funcionales —en especial el rendimiento (RNF-01)— se precisaron tras medir versiones preliminares.

El sistema tiene un único actor central, el **operador**: un técnico municipal —o un usuario no especialista— que compone y observa escenarios *what-if* sobre la red real. Sus casos de uso son seis: importar o cambiar el mapa (OSM), controlar la simulación, generar y reenrutar vehículos, componer eventos (incidentes, carriles, semáforos y ZBE), consultar métricas y analíticas, y visualizar e inspeccionar en 3D. Los de *composición de escenario* y *control* son los que habilitan el análisis *what-if* que motiva el sistema.

2.4.2 Priorización (MoSCoW) y criterios de aceptación

El *backlog* se gestionó mediante una estricta priorización (§ 4.2) basada en el método MoSCoW [CB94], que clasifica cada requisito en una de cuatro categorías según su criticidad para una entrega viable: *Must have* (esencial), *Should have* (importante), *Could have* (deseable) y *Won't have* (excluído del ciclo actual). Aunque finalmente todos los requisitos fueron implementados, esta evaluación determinó el orden de las iteraciones y el alcance de las entregas bajo la presión temporal del proyecto (§ 5.8). Las categorías se estructuraron de la siguiente manera:

- **Esenciales (*Must*):** Conforman el núcleo del gemelo digital. Incluyen la ingesta de datos, simulación y rutas (RF-01 a RF-03), la ZBE (RF-07), la comunicación (RF-08) y la visualización (RF-10). En el ámbito no funcional, recogen el rendimiento (RNF-01) y el tiempo real (RNF-02).
- **Importantes (*Should*):** Agrupan funcionalidades operativas clave como el control de la red, la gestión de eventos y el cálculo de métricas (RF-04 a RF-06 y RF-09). También engloban la modularidad (RNF-03) y la portabilidad (RNF-04).
- **Deseables (*Could*):** Se limitan exclusivamente a aportar observabilidad al sistema (RNF-05).

Los criterios de aceptación para dichos requisitos varían en función de su naturaleza. Ciertos objetivos son estrictamente cuantificables, caso del rendimiento (RNF-01), que exige mantener la simulación a $3\,\mathrm{Hz}$ con fluctuaciones de $4\,000$ a $6\,000$ vehículos (§ 6.6). Por el contrario, para los atributos cualitativos se han adoptado criterios estructurales explícitos; por ejemplo, la modularidad (RNF-03) se verifica asegurando un grafo de dependencias arquitectónicas puramente *acíclico*.

2.4.3 Matriz de trazabilidad de requisitos

Cada requisito se realiza en una sección de diseño (Capítulo 5) y se comprueba en una sección de validación (Capítulo 6). El Cuadro 2.4 hace explícita esa correspondencia requisito → diseño → validación. Las restricciones RES-01–RES-05 no figuran en la matriz por ser condicionantes de alcance, no funcionalidad verificable. Su efecto se discute en la § 5.1.1 y entre las limitaciones del Capítulo 7.

ID	Requisito (resumido)	Diseño	Validación
RF-01	Ingesta OSM (ETL)	§ 5.2.3	§ 6.1
RF-02	Microsimulación IDM/MOBIL	§ 5.3	§ 6.2
RF-03	Cálculo de rutas A*	§ 5.3.5	§ 6.3
RF-04	Ciclo de vida de la simulación	§ 5.4.1	§ 6.1
RF-05	Señalización y prioridades	§ 5.4.2, § 5.4.3	§ 6.4
RF-06	Incidentes y cierre de carriles	§ 5.4.4	§ 6.9
RF-07	Zonas restringidas (ZBE)	§ 5.4.5	§ 6.8
RF-08	API REST + canal WS	§ 5.5.1, § 5.5.2	§ 6.1
RF-09	Métricas de tráfico	§ 5.5.3	§ 6.5
RF-10	Visualización 3D interactiva	§ 5.5.4	§ 6.7
RNF-01	Rendimiento (3 Hz, 4–6k veh.)	§ 5.4.1	§ 6.6
RNF-02	Tiempo real (60 FPS)	§ 5.5.4	§ 6.7
RNF-03	Modularidad (capas)	§ 5.1.2	§ 6.1
RNF-04	Portabilidad (Docker)	§ 5.1.4	demostración (§ 5.7)
RNF-05	Observabilidad	§ 5.6.2	§ 6.6

Cuadro 2.4: Trazabilidad de cada requisito funcional y no funcional a su sección de diseño (Capítulo 5) y a su sección de validación (Capítulo 6)

Capítulo 3

Antecedentes y estado del arte

La construcción de un gemelo digital de tráfico urbano se apoya en cuatro pilares técnicos más o menos solapados: la *microsimulación vehicular*, las *plataformas de simulación* ya consolidadas, la *información geográfica abierta* sobre la que se reconstruye la red y el *cálculo de rutas* que guía a los vehículos por ella. Sobre estos se superpone un quinto eje, *el normativo*, dictado por las políticas europeas y españolas de movilidad sostenible y zonas de bajas emisiones. Este capítulo recorre los cinco ejes y termina con una síntesis crítica que sitúa el presente TFG frente al trabajo previo.

3.1 Microsimulación vehicular: niveles de modelado

La literatura clásica clasifica los modelos de tráfico en tres escalas de detalle crecientes:

- **Nivel macroscópico:** Analiza el tráfico como un fluido continuo mediante magnitudes globales o agregadas, como la densidad ρ, el flujo q y la velocidad media $\bar{v}$. Destacan como referentes clásicos el modelo fundacional de Lighthill-Whitham-Richards (LWR) [LW55] y el posterior Cell Transmission Model (CTM) [Dag95].

- **Nivel mesoscópico:** Supone un punto intermedio donde los vehículos se agrupan y simulan en pelotones para reducir la carga computacional; el enfoque de Newell [New02] es su exponente más conocido.

- **Nivel microscópico:** Simula cada vehículo como una entidad independiente con sus propias variables de estado. Es la única escala capaz de capturar cómo las decisiones individuales, los cambios de carril y los eventos puntuales (semáforos o incidentes) afectan a la circulación.

La elección de escala no es neutra: solo el nivel microscópico preserva la identidad de cada vehículo [TK13], condición necesaria para aplicar medidas que discriminan coche a coche, como una restricción de ZBE según la etiqueta ambiental [Gob22]. Por ello, los gemelos digitales de tráfico urbano orientados a escenarios *what-if* operan en esta escala; los apartados siguientes repasan sus tres modelos de comportamiento: el seguimiento vehicular, el cambio de carril y el arbitraje en intersecciones.

3.1.1 Modelos de seguimiento vehicular

La idea fundacional del seguimiento vehicular (*car-following*) consiste en modelar la aceleración de un vehículo en función del estado del que circula delante. El modelo de referencia desde el año 2000 es el IDM de Treiber, Hennecke y Helbing [THH00], que expresa la aceleración longitudinal como la suma de dos términos: uno de *velocidad libre*, que empuja al vehículo hacia su velocidad deseada, y otro de *interacción*, que lo frena conforme se acerca al de delante:

$$a_{\text{IDM}}(v, s, v_{\text{lead}}) \;=\; a\left[1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s^*(v, \Delta v)}{s}\right)^2\right] \tag{3.1}$$

donde la *distancia deseada* s^* —el hueco que el conductor querría mantener con el líder— encapsula su prudencia:

$$s^*(v, \Delta v) \;=\; s_0 \,+\, \text{máx}\!\left(0,\; v\,T \,+\, \frac{v\,\Delta v}{2\sqrt{a\,b}}\right) \tag{3.2}$$

El modelo evalúa el entorno a partir de tres variables dinámicas de entrada: la velocidad propia v, el espaciado respecto al líder s y la velocidad relativa respecto al líder $\Delta v = v - v_{\text{lead}}$.

A su vez, su comportamiento se rige por seis parámetros. Tres de ellos definen la prudencia y el objetivo de marcha: la velocidad libre deseada v_0, el intervalo temporal de seguridad T y la distancia mínima de parada s_0. Los otros tres definen la dinámica del vehículo: la aceleración máxima cómoda a, la deceleración cómoda b y un exponente δ (típicamente fijado en 4) que suaviza la aceleración a medida que $v \to v_0$.

Es precisamente esta formulación la que explica la adopción generalizada del modelo [TK13]. Destaca por tres motivos: la facilidad para calibrar e interpretar físicamente todos sus parámetros, su probada capacidad para reproducir fenómenos colectivos como las ondas *stop-and-go*, y su garantía de evitar colisiones en dominios continuos bajo una discretización del tiempo adecuada. Hoy es el modelo de seguimiento de referencia: Simulation of Urban MObility (SUMO) [LBBW$^+$18] lo incluye entre sus modelos disponibles, mientras que MATSim [HNA16] opera a otro nivel de detalle, con un modelo de colas por tramo.

3.1.2 Modelos de cambio de carril

El seguimiento vehicular solo cubre la dinámica longitudinal dentro del carril; la incorporación, la salida y el adelantamiento exigen además decidir *cuándo* cambiar de carril, combinando el *deseo* de cambiar con la *posibilidad* de hacerlo con seguridad.

El modelo dominante en la actualidad es el MOBIL —*Minimizing Overall Braking Induced by Lane changes*— de Kesting, Treiber y Helbing [KTH07]. Su idea central es no formular reglas propias, sino *reutilizar* el modelo de seguimiento: para cada vehículo implicado,

MOBIL pregunta al IDM qué aceleración tendría *antes* y *después* de un hipotético cambio y decide a partir de esa comparación. La Figura 3.1 fija la notación: c es el vehículo que se plantea el cambio (el *ego*), n su nuevo seguidor en el carril destino y o su antiguo seguidor; a_x denota la aceleración IDM actual del vehículo x y $\tilde{a}_x$ la que tendría tras el cambio.

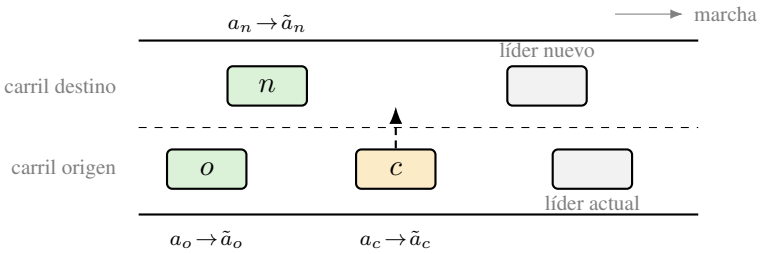

Figura 3.1: El vehículo c (*ego*, en naranja) evalúa desplazarse al carril de destino. Para decidir, el modelo evalúa con el IDM a los tres vehículos afectados por la maniobra: el propio c y los seguidores de ambos carriles (o y n, en verde), comparando sus aceleraciones antes (a_x) y después ($\tilde{a}_x$) del cambio. Los vehículos predecesores (en gris) no ven alterado su estado. El cambio solo procede si las nuevas aceleraciones cumplen los criterios de seguridad e incentivo (Ecuaciones (3.3) y (3.4)).

Sobre esta notación, MOBIL acepta el cambio si y solo si se cumplen dos condiciones. La primera es un **criterio de seguridad**: el nuevo seguidor no debe verse obligado a frenar más fuerte que un límite tolerable b_{safe},

$$\tilde{a}_n \;\geq\; -\,b_{\text{safe}} \tag{3.3}$$

lo que descarta de entrada las maniobras que provocarían una frenada brusca —o una colisión— en el carril destino. La segunda es un **criterio de incentivo**, que sopesa la ventaja del cambio frente a la molestia que ocasiona a los demás:

$$(\tilde{a}_c - a_c) \;+\; p\,\big[\,(\tilde{a}_n - a_n) + (\tilde{a}_o - a_o)\,\big] \;>\; \Delta a_{\text{th}} \tag{3.4}$$

El primer sumando es la ganancia de aceleración del propio *ego* al cambiarse, y el término entre corchetes, el efecto sobre los seguidores afectados, ponderado por el *factor de cortesía* $p \in [0, 1]$ (de 0, conductor egoísta, a 1, plenamente altruista). El *umbral* Δa_{th} descarta los cambios marginales y evita el zigzagueo entre carriles. La parametrización se reduce, pues, a tres magnitudes: la cortesía p, el frenazo seguro b_{safe} y el umbral Δa_{th}.

La virtud de esta formulación es que MOBIL no calcula aceleraciones por sí mismo, sino que reutiliza la fórmula del IDM para estimar la aceleración de cada vehículo antes y después del cambio. Actúa así como una capa de decisión sobre el modelo de seguimiento.

3.1.3 Arbitraje en intersecciones

El arbitraje en intersecciones no señalizadas y rotondas recibe un tratamiento solo tangencial en la literatura microscópica clásica. La aproximación habitual es la *aceptación de*

huecos (*gap-acceptance*) [TK13]: el vehículo decide entrar al cruce si el hueco espacial frente al primer vehículo prioritario excede un umbral en función de su propia velocidad y aceleración. Existen variantes que sustituyen los umbrales espaciales por umbrales temporales basados en el Time-To-Collision (TTC), el tiempo que tardarían en colisionar dos vehículos si mantuviesen su velocidad relativa: $\mathrm{TTC} = s/\Delta v$, para un espaciado neto s y una velocidad de acercamiento $\Delta v > 0$ (si $\Delta v \leq 0$ los vehículos no convergen y no existe riesgo). El cruce se concede cuando ese tiempo supera un umbral de seguridad: cuanto mayor es el TTC, más holgado es el hueco disponible.

3.2 Plataformas de microsimulación de referencia

Al diseñar un gemelo digital se debe tomar una decisión temprana entre incorporar una plataforma existente o reimplementar el motor a medida. Las cuatro alternativas dominantes son SUMO [LBBW+18, KEBB12], MATSim [HNA16] y las comerciales AIMSUN Next [CFG+10] y PTV VISSIM [FV10].

Estas cuatro opciones difieren en dos ejes decisivos: la licencia y la granularidad del modelado. AIMSUN Next y PTV Verkehr In Städten – SIMulationsmodell (VISSIM) son productos comerciales de licencia cerrada, orientados al mercado de la consultoría de tráfico. MATSim, aun siendo libre, no es un microsimulador puro: modela la demanda como una secuencia de actividades diarias y su resolución espacio-temporal no alcanza el nivel subsegundo (*tick*) al que se actualiza la dinámica vehicular individual. SUMO queda, así, como la única plataforma a la vez libre y verdaderamente microscópica del panorama actual.

Mención aparte merece su ecosistema de integración. Su interfaz Traffic Control Interface (TraCI) permite gobernar la simulación paso a paso —consultar y modificar vehículos, semáforos o carriles en cada paso— y sostiene desde hace años acoplamientos en línea consolidados, como el de *Veins* con simuladores de redes de comunicación vehicular [SGD11]. Un lazo interactivo sobre SUMO es, por tanto, técnicamente viable; elegir entre integrar una plataforma así o implementar un motor a medida depende ya de los requisitos de cada proyecto.

3.3 Información geográfica: representación, fuentes y almacenamiento

Una red vial es, antes que un grafo de simulación, un conjunto de *información geográfica* -nodos, tramos y áreas- anclados a coordenadas sobre la superficie terrestre. Reconstruir un escenario realista obliga, por tanto, a resolver tres cuestiones encadenadas, que esta sección recorre en orden: *cómo* se representa esa información de forma que una máquina pueda razonar sobre ella, *de dónde* se obtiene y *dónde* se almacena para consultarla con eficiencia.

3.3.1 Representación: el modelo vectorial y *Simple Features*

El estándar *Simple Features* de Open Geospatial Consortium (OGC) [Ope10] define los tipos geométricos canónicos (`POINT`, `LINESTRING`, `POLYGON`) y los operadores espaciales (`ST_Intersects`, `ST_Distance`, entre otros) que la mayoría de motores espaciales implementan. Esos tres tipos son el vocabulario que una red vial necesita: una intersección, un semáforo o la posición instantánea de un vehículo se modelan como `POINT`; el trazado de un tramo de calle, como `LINESTRING`; y el perímetro de una ZBE, como `POLYGON`. Almacenar el dominio con estos tipos hace que el motor de base de datos «entienda» la geometría, en vez de tratarla como columnas numéricas opacas sobre las que toda la lógica espacial recaería en la aplicación.

De los operadores, dos familias clásicas concentran el uso sobre una red vial: la *topológica* (`ST_Intersects`, `ST_Contains`), que determina qué aristas caen dentro del polígono de una zona, y la *métrica* (`ST_Distance`), que ancla una coordenada Global Positioning System (GPS) al nodo más cercano. Como predicados Structured Query Language (SQL) declarativos, ambas son acelerables con un índice espacial sin reescribir la consulta, y la semántica y serializaciones comunes del estándar permiten que una misma geometría circule sin conversiones *ad hoc* entre la base de datos, el código de aplicación y cualquier Geographic Information System (GIS) de escritorio.

3.3.2 Fuentes de datos cartográficos abiertos: OpenStreetMap

Las fuentes de cartografía digital se agrupan en tres familias: la oficial de los organismos cartográficos, la comercial de los proveedores de mapas y la abierta colaborativa. Esta última tiene en OpenStreetMap (OSM) a su máximo exponente. Es la fuente abierta de datos cartográficos con cobertura mundial más completa [HW08]. Su modelo de datos es deliberadamente simple y reutiliza las primitivas vectoriales ya descritas: *nodes* (puntos), *ways* (polilíneas o polígonos) y *relations* (agrupaciones), todos ellos anotados con *etiquetas* clave/valor que describen su naturaleza (`highway=residential`, `maxspeed=50`, `junction=roundabout...`). Esa flexibilidad, una licencia abierta que permite redistribuir y transformar los datos, y un ecosistema de herramientas de extracción y procesado ya consolidado [RTC10] explican que OSM se haya convertido en la fuente de referencia frente a sus dos alternativas: evita las restricciones de licencia y los costes por uso de los proveedores comerciales, y ofrece una cobertura y una frescura que la cartografía oficial no siempre garantiza.

A cambio de su apertura, OSM hereda la heterogeneidad propia de un proyecto colaborativo: la calidad es alta para las vías estructurantes pero más irregular en el viario secundario [HW08], donde pueden faltar atributos como el número de carriles o el etiquetado de los semáforos. El procedimiento habitual es aplicar *valores por defecto* sensatos cuando una *etiqueta* está ausente, siguiendo recomendaciones consolidadas como las de Ramm y Topf [RTC10].

3.3.3 Almacenamiento y consulta espacial: PostGIS

Entre las implementaciones abiertas de *Simple Features*, PostGIS —la extensión geoespacial de PostgreSQL [OH21]— es la referencia. Ofrece el soporte más completo del estándar entre los motores libres y, sobre todo, índices espaciales Generalized Search Tree (GiST) [Gut84] que llevan las consultas espaciales a coste logarítmico. Frente a sus alternativas, aúna ser un servidor concurrente (a diferencia de SpatiaLite), de licencia abierta (frente a Oracle o SQL Server) y heredar el motor transaccional maduro de PostgreSQL, lo que permite tratar la geometría como una columna más del modelo relacional.

Cualquier almacenamiento espacial debe fijar, por último, un sistema de referencia de coordenadas. El más extendido para datos vinculados a coordenadas GPS es World Geodetic System 1984 (WGS84), de código European Petroleum Survey Group (EPSG) 4326, que expresa las posiciones en latitud y longitud. Esas unidades angulares, sin embargo, no son cómodas para los cálculos métricos (distancias o longitudes de tramo), que requieren proyectar a un plano cartesiano. A escala continental se recurre a proyecciones como Universal Transverse Mercator (UTM) o Web Mercator (EPSG 3857); pero a escala urbana, donde la región cubierta es pequeña, basta una aproximación *equirrectangular* [Sny87, Ven22] centrada en el centroide de la red, que la trata como un plano y convierte cada coordenada a metros mediante dos factores de escala constantes:

$$\Delta x \approx k\,(\lambda - \lambda_0)\cos\varphi_0, \qquad \Delta y \approx k\,(\varphi - \varphi_0), \qquad k \approx 111\,320 \text{ m/grado} \tag{3.5}$$

donde (λ, φ) son la longitud y la latitud del punto, (λ_0, φ_0) las del centroide de la red y k los metros que abarca aproximadamente un grado sobre un meridiano. El coseno de la latitud corrige el progresivo estrechamiento de los meridianos al alejarse del ecuador. Al fijarlo una sola vez para toda la red, en lugar de recalcularlo punto a punto, la conversión se reduce a una resta y un producto. La combinación natural a escala urbana es, por tanto, WGS84 para el intercambio de datos y una proyección local barata para el cómputo.

3.4 Cálculo de rutas sobre redes viales

El cálculo de la ruta más corta entre dos nodos de una red dirigida y ponderada es uno de los problemas más estudiados de las ciencias de la computación. El algoritmo de Dijkstra [Dij59] lo resuelve en $\mathcal{O}((|V|+|E|)\log|V|)$ con una cola de prioridad, explorando los nodos en orden creciente de coste desde el origen. La heurística de A* [HNR68] acelera esa búsqueda cuando se dispone de una estimación *admisible* del coste real al destino. Sobre una red geográfica esa cota es inmediata: si el coste que se minimiza es el tiempo de viaje, la distancia euclídea al destino dividida por la máxima velocidad alcanzable lo acota siempre por debajo,

$$h(u) \;=\; \frac{d_{\text{euc}}(u,\,\text{destino})}{v_{\text{máx}}} \;\leq\; h^{*}(u) \tag{3.6}$$

donde $h^{*}(u)$ es el tiempo real mínimo desde el nodo u hasta el destino (ningún trayecto puede ser más corto que la línea recta ni recorrerse a más de $v_{\text{máx}}$ [BDG+16]). Es precisamente esa admisibilidad la que garantiza que A* devuelva siempre un camino óptimo, a la vez que poda gran parte del espacio de búsqueda que Dijkstra exploraría a ciegas.

3.5 Movilidad sostenible y zonas de bajas emisiones

La movilidad sostenible es el quinto eje de estos antecedentes, no técnico sino normativo y conceptual. Banister [Ban08] formalizó el *paradigma de la movilidad sostenible* como un giro desde la maximización del flujo motorizado hacia la maximización del acceso a oportunidades. Las *supermanzanas* de Barcelona, analizadas cuantitativamente por Mueller *et al.* [MRRK+20], son una materialización urbanística reciente del paradigma.

3.5.1 La normativa de tráfico como entrada de la simulación

En España, el Reglamento General de Circulación [Gob03] fija el comportamiento esperado del conductor, y esa regulación es precisamente el conjunto de entradas que parametriza una microsimulación fiel. Dos familias de normas resultan determinantes para los modelos microscópicos de la § 3.1.

La primera es la **limitación de velocidad**. La velocidad genérica urbana es de 50 km/h, pero la reforma del Real Decreto 970/2020 [Gob20] rebajó el límite a 30 km/h en las vías de un único carril por sentido y a 20 km/h en las de plataforma única. Estos topes legales se trasladan directamente al parámetro de *velocidad libre deseada* v_0 del IDM (Ecuación (3.1)).

La segunda son las **normas de precedencia**. La prioridad a la derecha en cruces no señalizados, las señales de ceda el paso y `stop`, y la cesión al anillo en las rotondas establecen quién pasa primero cuando dos trayectorias compiten por el mismo punto. Un gemelo digital que las ignorase produciría trayectorias físicamente plausibles pero ilegales, inservibles para razonar sobre medidas de movilidad.

Sobre esta capa de reglas se superpone un segundo nivel regulatorio, más reciente y más disruptivo para el reparto modal, que actúa no ya sobre cómo se conduce sino sobre qué vehículos pueden circular: las zonas de bajas emisiones.

3.5.2 Zonas de bajas emisiones

En el plano normativo español, dos instrumentos articulan las ZBE:

Ley 7/2021 de cambio climático y transición energética [Cor21], cuyo artículo 14 obliga a los municipios de más de 50 000 habitantes a establecer una Zona de Bajas Emisiones antes del 1 de enero de 2023.

Real Decreto 1052/2022 [Gob22] que regula los requisitos mínimos de las ZBE: perímetro, restricciones por etiqueta ambiental de la Dirección General de Tráfico (DGT), sistema de control y régimen sancionador.

A diferencia de un límite de velocidad, que afecta por igual a cualquier vehículo en un tramo, una ZBE discrimina coche a coche según su etiqueta ambiental de la DGT. Es justamente esta granularidad la que exige el nivel microscópico —no se puede filtrar un pelotón homogéneo (§ 3.1)— y la que convierte a la ZBE en el escenario *what-if* más natural para un gemelo digital por vehículo.

Los efectos de las ZBE sobre la calidad del aire han sido objeto de varios estudios empíricos. Salas *et al.* [SPVPRR21] cuantificaron el impacto de Madrid Central sobre los niveles de NO_2, encontrando reducciones significativas dentro del perímetro pero efectos rebote moderados en las arterias circundantes. Esa literatura motiva la simulación *antes* de implantar la ZBE. Las herramientas de gemelo digital pueden ayudar a anticipar los efectos rebote y a dimensionar las medidas compensatorias.

3.5.3 El caso de Talavera de la Reina

Talavera de la Reina (Toledo, $\sim 84\,000$ habitantes) entra en el grupo de municipios obligados por la Ley 7/2021. Su trayectoria reciente concentra varias actuaciones y revisiones en poco tiempo:

- **Aforos.** Las grandes vías de acceso cuentan con aforos del Ministerio de Transportes, Movilidad y Agenda Urbana (MITMA), pero las intensidades urbanas por calle no se publican abiertamente, pese a haberlas medido el Plan de Movilidad Urbana Sostenible (PMUS) de 2023 [Doy23, Min24a].
- **ZBE.** Aprobada inicialmente en octubre de 2025 [Ayu25] y *aplazada* en marzo de 2026 por defectos en el trámite de información pública [CLM26], con la infraestructura de control ya desplegada [Cov24]. Se concibió con carácter *no sancionador*: inactiva salvo superarse los umbrales de calidad del aire [Esp24].
- **Otras actuaciones.** En paralelo conviven obras de humanización y desdoblamiento de viario y episodios como los cortes por la crecida del Tajo, que mantienen la movilidad de la ciudad en cambio constante [Min24a].

Esta combinación de actualidad, datos parcialmente disponibles y decisiones políticas activas hace de Talavera un caso de estudio particularmente adecuado para un gemelo digital descriptivo.

3.6 Síntesis: hueco que aborda este TFG

La revisión anterior permite delimitar el *hueco* que aborda el TFG. No falta literatura sobre IDM/MOBIL [THH00, KTH07], ni plataformas de código abierto de microsimula-

ción [LBBW+18, HNA16]. Sí falta —en el conocimiento de quien escribe— una pieza pequeña y concreta:

Un simulador de código abierto que combine ingesta directa de OSM sobre PostGIS, IDM/MOBIL parametrizados para entorno urbano español, gestión de ZBE configurable y visualización 3D interactiva a 60 Frames Per Second (FPS), todo ello con código de tamaño abarcable por una sola persona y diseñado para escenarios *what-if* sobre ciudades medianas españolas.

Situar con precisión ese hueco exige evaluar las herramientas existentes no por su calidad absoluta, sino por su *ajuste* a los seis requisitos que definen este TFG: **(R1)** resolución microscópica por vehículo con *tick* subsegundo; **(R2)** carácter de código abierto y reproducible; **(R3)** ingesta directa de OSM sobre una base de datos espacial (PostGIS); **(R4)** discriminación coche a coche para gestionar la ZBE (por tipo de vehículo en el prototipo, extensible a la etiqueta ambiental de la DGT); **(R5)** visualización 3D interactiva a 60 FPS alimentada en tiempo real; y **(R6)** una base de código abarcable y modificable por una sola persona. El Cuadro 3.1 resume cómo se proyecta cada herramienta sobre estos ejes con un criterio uniforme.

Herramienta	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**
SUMO+TraCI	●	●	○	○	–	–
MATSim	–	●	○	–	–	–
AIMSUN / VISSIM	●	–	○	○	●	–
CARLA	●	●	–	–	○	–
Cesium / deck.gl	–	●	○	–	●	○
Este TFG	●	●	●	●	●	●

Cuadro 3.1: Cobertura de los seis requisitos del TFG por las herramientas de referencia. ● cubre, ○ parcial, – no cubre [LBBW+18, HNA16, CFG+10, FV10, DRC+17, Ces24, vO24].

La lectura por filas confirma que ninguna herramienta cubre los seis ejes a la vez. CARLA (*Car Learning to Act*) [DRC+17] está pensado para vehículos autónomos, no para políticas de movilidad; Cesium o deck.gl [Ces24, vO24] solo resuelven la visualización, sin motor de simulación detrás; MATSim y las plataformas comerciales tropiezan con la granularidad o con la apertura. La alternativa más cercana es SUMO [LBBW+18], que aporta el modelo microscópico nativo, la licencia libre y un lazo interactivo técnicamente viable (§ 3.2).

Tres consideraciones de integración justifican, aun así, optar por un motor propio. La primera es el **modelo de datos** (R3): si la red debe vivir en una base de datos espacial como información geográfica estándar, el formato interno de SUMO obligaría a mantener dos representaciones de la misma red y la correspondencia de identificadores entre ambas. La segunda es la **semántica del control** (R4): una ZBE «no sancionadora» exige penalizar rutas sin prohibirlas y ajustar los pesos al estado vivo de la red, es decir, control directo sobre el cálculo de rutas; las restricciones por clase de vehículo de SUMO, en cambio, actúan como

permisos que vetan el arco. Y la tercera es el **alcance** (R6): aun con SUMO como núcleo habría que construir alrededor casi todo el sistema (capa de servicio, analíticas y cliente de visualización), extrayendo y reinyectando su estado completo en cada *tick*, sobre una base de código cuyo tamaño la hace inabarcable como punto de partida para una sola persona.

El valor del proyecto no está, pues, en superar a SUMO en microsimulación ni a CARLA en realismo gráfico, sino en *integrar* en una sola pieza la cadena completa, llenando el espacio intermedio para escenarios docentes, de divulgación y de *prototipado rápido* de medidas de movilidad sobre ciudades como Talavera de la Reina.

Capítulo 4

Metodología

ANTES de describir qué se construyó conviene explicar cómo se construyó. Este capítulo detalla el proceso de desarrollo que ha vertebrado el TFG. En primer lugar, se expone la metodología base: el marco de trabajo iterativo e incremental (§ 4.1), apoyado en el uso de repositorios para garantizar la organización y trazabilidad del proyecto (§ 4.2). A continuación, se desglosa la planificación temporal mediante iteraciones (§ 4.3) y se enumeran las herramientas de software y hardware empleadas (§ 4.4). La segunda mitad del capítulo aborda la dimensión reflexiva del desarrollo, en particular la gestión de riesgos del proyecto (§ 4.6). El resultado de aplicar esta metodología se abordará en el Capítulo 5 —donde también se recogen las incidencias y desviaciones respecto al plan y la retrospectiva del proceso—, mientras que su validación empírica se reserva para el Capítulo 6.

4.1 Proceso de desarrollo: iterativo e incremental

El desarrollo siguió un **proceso iterativo e incremental**, en el espíritu de las metodologías ágiles [BBv+01]: el sistema se construyó en iteraciones sucesivas, cada una con un objetivo acotado y cerrada con un **incremento ejecutable y demostrable**. De los marcos ágiles se conservaron las prácticas que aportan valor a esta escala: el **backlog** de tareas priorizado y materializado como *issues* del repositorio, y la **revisión** al cierre de cada iteración que alimentaba la planificación de la siguiente. Se descartó la adpoción de marcos formales como Scrum, puesto que sus roles diferenciados (*Scrum Master*, *Product Owner*) y prácticas habituales (como *dailys*) asumen un trabajo en equipo. En su lugar, se priorozó la flexibilidad y el avance del proyecto individual, en lugar de la burocracia formal del proceso.

La elección de este enfoque —frente a un modelo en cascada— responde a tres características del proyecto: su **amplitud** (el sistema integra piezas muy heterogéneas cuyo encaje no podía especificarse por completo de antemano), el **riesgo técnico** de esa integración (que aconsejaba validar cada pieza sobre un sistema funcionando antes de añadir la siguiente) y su condición de **prototipo exploratorio** (muchas decisiones, como la frecuencia de simulación o el formato de transmisión, solo podían tomarse tras medir una versión preliminar). El carácter incremental permitió, además, disponer en todo momento de una versión funcional que mostrar y sobre la que razonar.

Cada iteración siguió el mismo ciclo: se seleccionaban del *backlog* las tareas de mayor prioridad y se abría un *issue* por cada una; se desarrollaban en una rama propia; se integraban mediante una *pull request* que servía de punto de revisión; y se cerraba la iteración con una versión ejecutable que se contrastaba con los objetivos parciales del Capítulo 2. El trabajo se repartió en **seis iteraciones** sucesivas, cuyo orden (infraestructura, datos, núcleo de simulación, control, cliente y rendimiento) es deudor del principio de que cada subsistema se apoya en los anteriores:

Iteración 1 — Infraestructura y arquitectura
: Esqueleto cliente-servidor, contenedorización del backend y estructura de los dos proyectos. Fija las decisiones arquitectónicas que condicionan todo lo demás (§ 5.1).

Iteración 2 — Datos y persistencia
: Modelo de datos geoespacial, proceso Extract, Transform, Load (ETL) de ingesta desde OSM y construcción del grafo vial sobre el que opera el simulador (§ 5.2).

Iteración 3 — Núcleo de simulación
: Modelo microscópico de tráfico —IDM, MOBIL— y cálculo de rutas con A* (§ 5.3).

Iteración 4 — Control y eventos
: Semáforos, prioridades en intersecciones, incidentes y la ZBE: los mecanismos que alteran el tráfico base (§ 5.4).

Iteración 5 — Cliente 3D y comunicación
: Canal de estado en tiempo real, motor de analíticas de tráfico y visualización 3D interactiva del tráfico simulado (§ 5.5).

Iteración 6 — Rendimiento y optimización
: Paralelización del bucle de simulación, reducción del grafo y formato binario de transmisión, para escalar el sistema a varios miles de vehículos (§ 5.6).

4.2 Gestión del trabajo con el repositorio

La organización del trabajo se apoyó de forma intensiva en *Git* y en la plataforma *GitHub*, que actuaron a la vez como sistema de control de versiones, tablero de planificación y registro de la trazabilidad del proyecto. El código y la documentación se repartieron en tres repositorios bajo una organización común (`DigitalTwinTalavera`): el backend de simulación en Python[1], el cliente 3D en Godot[2] y la propia memoria en LaTeX. La organización del trabajo se apoyó de forma intensiva en *Git* y en la plataforma *GitHub*, que actuaron a la vez como sistema de control de versiones, tablero de planificación y registro de la trazabilidad del proyecto. El código y la documentación se repartieron en tres repositorios bajo una organización común

[1] `https://github.com/DigitalTwinTalavera/tfg-dt-python-backend`
[2] `https://github.com/DigitalTwinTalavera/tfg-dt-godot-client`

(`DigitalTwinTalavera`): el backend de simulación en Python[3], el cliente 3D en Godot[4] y la propia memoria en LaTeX.

Flujo de ramas. Se adoptó una variante ligera de *git-flow* [Dri10]: una rama `main` estable, una rama `develop` de integración y una **rama de *feature* por cada tarea**, lo que aísla el trabajo en curso y mantiene `develop` siempre ejecutable. Las iteraciones de mayor calado tecnológico recibieron además ramas temáticas de larga duración (como `mejora-rendimiento` para la sexta iteración), sobre las que se acumularon varias mejoras antes de integrarlas en bloque.

Issues como *backlog*. Cada unidad de trabajo se registró como un *issue*, y cada rama de *feature* se nombró anteponiendo el número del *issue* a una descripción breve de la tarea (por ejemplo, `10-26-openstreetmap-data-loader-etl-pipeline`, correspondiente a la ingesta ETL desde OSM). Esta convención enlaza de forma automática y verificable cada rama con la tarea que la motivó, de modo que el conjunto de *issues* constituye el *backlog* priorizado del proyecto y su historia documenta qué se hizo en cada momento. El Cuadro 4.1 resume cómo se distribuyeron las tareas principales entre las iteraciones.

Iteración	**Tareas principales (*issues*)**
1 — Infraestructura	Inicialización del backend y API; infraestructura PostgreSQL/PostGIS; arranque del proyecto Godot y cliente Hypertext Transfer Protocol (HTTP)
2 — Datos y persistencia	Modelos de red vial; capa de repositorios; constructor del grafo; *cargador* ETL de OSM; carga de la red en el cliente
3 — Núcleo de simulación	Bucle de simulación con máquina de estados; generador de vehículos y asignación de rutas; modelos físicos IDM/MOBIL
4 — Control y eventos	Semáforos, prioridades e incidentes; correcciones de carriles y de la lógica de intersecciones
5 — Cliente y comunicación	Gestor de conexiones WS; difusión de estado en tiempo real; *renderizadores* 3D; interpolación de posiciones
6 — Rendimiento	Paralelización por hilos; reducción de nodos del grafo; formato binario de transmisión

Cuadro 4.1: Reparto de las tareas principales del *backlog* entre las seis iteraciones, según los *issues* y las ramas del repositorio

Integración y trazabilidad. La *pull request* que fusiona cada rama en `develop` fue el punto de revisión del trabajo: en ella se contrastaba el incremento con el objetivo del *issue* antes de incorporarlo. Junto con los mensajes de *commit*, este flujo deja una traza que enlaza objetivos, tareas, código y secciones de esta memoria.

[3] `https://github.com/DigitalTwinTalavera/tfg-dt-python-backend`

[4] `https://github.com/DigitalTwinTalavera/tfg-dt-godot-client`

4.3 Planificación temporal

El desarrollo del sistema se concentró en el primer semestre de 2026. La Figura 4.1 resume su planificación temporal mediante un diagrama de Gantt que sitúa las seis iteraciones y la fase de redacción sobre el calendario real, reconstruido a partir del historial del repositorio.

El proyecto arrancó a mediados de enero con una fase de construcción intensa: en apenas cinco semanas se levantaron la infraestructura, la capa de datos, el núcleo de simulación y una primera versión del cliente, de forma marcadamente solapada. Tras este empuje inicial (enero y febrero), marzo fue un periodo de baja actividad de desarrollo. En abril se retomó el trabajo con la sexta iteración, dedicada por completo al rendimiento (paralelización del bucle de simulación, reducción del grafo y formato binario de transmisión) sobre la rama `mejora-rendimiento`, y al refinamiento de la lógica de control. Finalmente, la redacción de esta memoria ocupó los meses de mayo y junio, en paralelo a los últimos ajustes del sistema.

La *línea base* prevista al inicio era el desarrollo *secuencial* de las seis iteraciones en el orden del Cuadro 4.1, con la redacción al final. El Gantt de la Figura 4.1 representa, en cambio, la ejecución *real* reconstruida del historial del repositorio; las desviaciones que documenta la § 5.8 se entienden respecto a aquella intención inicial.

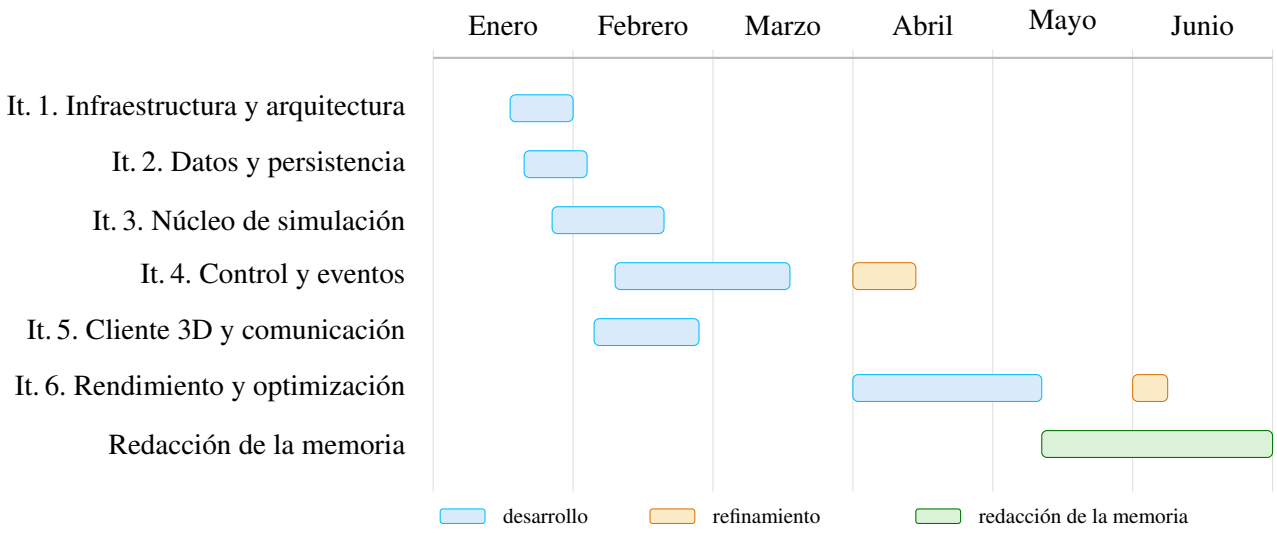

Figura 4.1: Diagrama de Gantt del proyecto. Las seis iteraciones se solapan durante la fase de construcción inicial (enero–febrero); la iteración de rendimiento se concentra en abril, y la redacción de la memoria, en mayo y junio

4.4 Herramientas

El desarrollo se apoyó en un conjunto de herramientas de código abierto, elegidas por su madurez y por encajar en una arquitectura cliente-servidor en tiempo real. Se enumeran a continuación, agrupadas en software de desarrollo y plataforma de ejecución y pruebas.

4.4.1 Software

Python 3.14 (*free-threaded*)

Lenguaje del backend. Se empleó la variante *free-threaded* (`3.14t`, Python Enhancement Proposal (PEP) 703), que desactiva el Global Interpreter Lock (GIL) y permite paralelismo real entre hilos, clave para la sexta iteración.

FastAPI

Framework web asíncrono sobre el que se expusieron la API REST de comandos y el canal WS de estado.

PostgreSQL + PostGIS

Base de datos relacional con extensión geoespacial; almacena la red vial y permite consultas espaciales nativas.

SQLAlchemy y GeoAlchemy2

Object-Relational Mapping (ORM) y su extensión geoespacial, que median entre el código y la base de datos.

Alembic

Gestión versionada del esquema de la base de datos mediante migraciones.

NetworkX

Construcción y consulta del grafo vial en memoria sobre el que opera el cálculo de rutas A*.

Godot 4.6

Motor de videojuegos, empleado como cliente de visualización 3D interactiva, con renderizado masivo mediante *MultiMesh*.

Docker y Docker Compose

Contenedorización y orquestación del backend y la base de datos, para un despliegue reproducible.

Overpass Turbo

Servicio de consulta sobre OSM con el que se descargaron los extractos cartográficos de entrada.

Prometheus

Formato e instrumentación de las métricas de rendimiento del backend.

pytest

Framework de pruebas automatizadas del backend.

Git y GitHub

Control de versiones y plataforma de organización del trabajo (§ 4.2).

LaTeX

Composición de esta memoria.

4.4.2 Hardware

El sistema se desarrolló y se midió sobre una única máquina: un procesador Intel Core i5-13600KF (14 núcleos físicos, 20 hilos lógicos, hasta 5,1 GHz), una tarjeta gráfica NVIDIA GeForce RTX 3070 Ti (8 GiB), 31 GiB de Random Access Memory (RAM) y sistema operativo Linux (*kernel* 6.19). Esta misma configuración es la que se emplea como banco de pruebas en las mediciones de rendimiento del Capítulo 6, por lo que las cifras allí reportadas deben interpretarse siempre en relación con este *hardware*.

4.5 Esfuerzo y presupuesto

No se fijó al inicio una estimación de esfuerzo cerrada, por lo que las cifras que siguen son una *reconstrucción* a partir de la actividad del repositorio, no una línea base contra la que medir desviaciones.

Iteración / fase	**Esfuerzo estimado (h)**
1 — Infraestructura y arquitectura	40
2 — Datos y persistencia	60
3 — Núcleo de simulación	95
4 — Control y eventos	70
5 — Cliente 3D y comunicación	85
6 — Rendimiento y optimización	90
Redacción de la memoria	80
Total	$\sim$ **520**

Cuadro 4.2: Esfuerzo de desarrollo estimado por iteración

Todo el coste recae en el tiempo de desarrollo, puesto que todo el *software* empleado es de código abierto y se desarrolló sobre un equipo personal ya disponible, por lo que el coste de licencias es *nulo*.

4.6 Gestión de riesgos

Un proyecto de esta amplitud, abordado por un único desarrollador y con un plazo académico fijo, convivió desde el inicio con un conjunto de riesgos identificables. El propio proceso iterativo es, en buena medida, una estrategia de gestión de riesgos, validando primero las piezas de mayor incertidumbre técnica sobre un sistema funcionando (§ 4.1). El Cuadro 4.3 recoge los seis riesgos principales, su valoración inicial —probabilidad, impacto y la *exposición* resultante $P \times I$, que los ordena por prioridad—, la estrategia de mitigación adoptada y el desenlace de cada uno.

De estos seis riesgos, cuatro acabaron materializándose en mayor o menor medida —R1, R4, R5 y, en parte, R6— y se tradujeron en las incidencias y desviaciones que documenta la sección siguiente. R2 y R3, en cambio, se mantuvieron controlados: la integración continua

Riesgo	P	I	P × I	Estrategia de mitigación	Desenlace
R1. Escalado del backend en Python, con el GIL como límite	Alta	Alto	9	Arquitectura apta para paralelizar desde el inicio; CPython *free-threaded* (§ 4.4.1); una iteración dedicada al rendimiento	Materializado
R2. Integración de subsistemas heterogéneos	Media	Alto	6	`develop` siempre ejecutable e integración continua por *pull requests*	Controlado
R3. Cadena de herramientas experimental (*free-threaded*)	Media	Medio	4	Repliegue (*fallback*) a un solo hilo: un fallo degradaría el rendimiento, no la corrección	Controlado
R4. Calidad de los datos (OSM ruidoso, sin aforos)	Alta	Medio	6	Limpieza en el ETL; tráfico sintético y validación cualitativa	Materializado
R5. Tiempo y dedicación (un desarrollador, plazo fijo)	Alta	Alto	9	*Backlog* priorizado y recorte deliberado del alcance no esencial	Materializado
R6. Corrección del modelo de tráfico (bloqueos, colisiones)	Media	Medio	4	Pruebas dirigidas a intersecciones; detección de tareas que bloquean el bucle	Parcial

Cuadro 4.3: Principales riesgos identificados al inicio, su valoración —probabilidad P e impacto I en escala Alta/Alto $= 3$, Media/Medio $= 2$; exposición $P \times I$—, la estrategia de mitigación y el desenlace. La exposición ordena los riesgos por prioridad (R1 y R5, los más expuestos, fueron en efecto los más determinantes); el desenlace se detalla en la § 5.8

evitó sorpresas de ensamblaje y el repliegue a ejecución monohilo nunca llegó a ser necesario, aunque permaneció disponible como red de seguridad durante todo el proyecto.

Capítulo 5

Resultados

ESTE capítulo presenta el resultado de aplicar el proceso de desarrollo descrito en el Capítulo 4. Mientras que el Capítulo 3 presentó las herramientas conceptuales —IDM, MOBIL, A*, *Simple Features*— en abstracto, aquí se llevan a la práctica, mostrando cómo se han ensamblado en un sistema concreto y por qué se han tomado las decisiones de diseño que lo articulan. La exposición recorre las seis iteraciones de construcción y culmina con el sistema completo en ejecución (§ 5.7).

El capítulo se estructura siguiendo el camino que recorren los datos: desde su origen hasta la pantalla. Este recorrido coincide cronológicamente con el orden en que se construyó el sistema (§ 4.3). La Figura 5.1 ilustra todo este ciclo de vida —desde la ingesta del fichero OSM hasta la generación del fotograma— que se detallará en las siguientes secciones. En paralelo a este *flujo descriptivo* opera el *flujo de control*: los comandos del operador, recibidos a través de la API REST, alteran la topología del grafo o el estado interno del motor, reflejándose en la siguiente instantánea de la simulación.

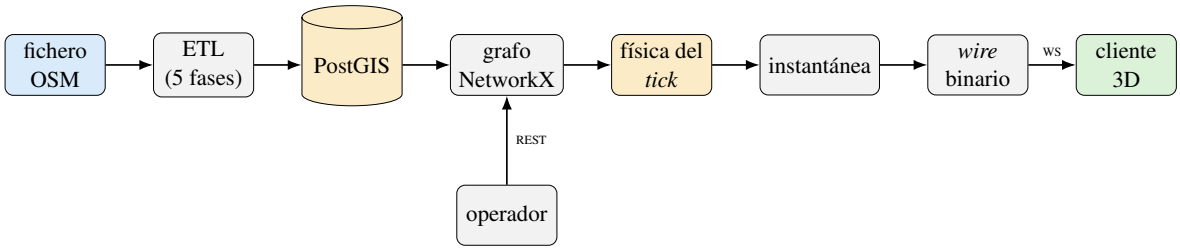

Figura 5.1: Linaje de los datos de extremo a extremo. *Camino descriptivo* (fila superior): el fichero OSM se ingiere por ETL a PostGIS, se reconstruye como grafo en memoria, lo anima la física del *tick*, se serializa en una instantánea binaria y se difunde por WS al cliente 3D. *Camino de control*: los comandos del operador entran por REST y mutan el grafo y el motor, reflejándose en la instantánea siguiente

5.1 Iteración 1: Infraestructura y arquitectura

La primera iteración tuvo como objetivo sentar las bases sobre las que crecería todo lo demás: una arquitectura clara y un esqueleto cliente-servidor desplegable, aunque todavía vacío de simulación (los dos proyectos en marcha, el backend contenedorizado y comunicándose con el cliente), lo que permitió validar desde el primer día las decisiones más

estructurantes.

5.1.1 Decisiones arquitectónicas globales

El sistema se concibe como un *gemelo digital descriptivo* en el sentido introducido en la § 1.2. Evoluciona en tiempo simulado y responde a los comandos de un operador, sin realimentarse desde el mundo físico. De esa naturaleza se desprende la primera decisión, la más estructurante de todas: separar la lógica de simulación de la presentación. Esta separación, que recomienda la literatura de arquitectura de software [BCK21] y que adoptan la mayoría de plataformas de microsimulación [LBBW+18, HNA16], se materializa en dos piezas independientes (un servidor que gobierna el modelo numérico y un cliente que se limita a visualizarlo y a enviar las acciones del usuario) y condiciona todo lo que sigue.

El resto de decisiones de alto nivel no se han tomado en el vacío, sino para satisfacer un conjunto de requisitos de calidad concretos. El Cuadro 5.1 enumera los seis *condicionantes* dominantes del proyecto, la decisión adoptada para cada uno y la alternativa razonable que se descartó.

Condicionante	**Decisión adoptada**	**Rechazada**
Rendimiento (escalable a 4 000 vehículos)	Bucle de *tick* en *asyncio* con hilos *free-threaded* (Python 3.14t) y agrupación espacial 8×8	*ProcessPoolExecutor* (descartado por coste de IPC)
Independencia de despliegue	Servidor en contenedor Docker, cliente como ejecutable Godot	Monolito acoplado
Tiempo real bajo coste de red	WS binario *little-endian* con multi-fragmento	gRPC, Server-Sent Events (SSE)
Persistencia geoespacial	PostgreSQL con extensión PostGIS	SpatiaLite, MongoDB
Visualización masiva 3D	Godot 4.6 con *MultiMesh* y *ImmediateMesh*	Unity, Unreal, Three.js
Reentrenable a otra ciudad	Carga dinámica de archivos OSM *XML*	Codificar Talavera de forma estática

Cuadro 5.1: Condicionantes arquitectónicos y decisiones tomadas, frente a alternativas razonables descartadas

La separación cliente-servidor obliga a definir un protocolo de comunicación de baja latencia. Se han adoptado dos canales complementarios: REST sobre HTTP [Fie00] para los comandos puntuales (arranque de la simulación, alta de una zona, creación de un incidente) y WS [FM11] para el flujo continuo de estado, con las posiciones de los vehículos a la frecuencia del *tick* (3 Hz por defecto). Esta dualidad renuncia a la simplicidad del canal único, pero permite explotar la caché y la semántica idempotente del HTTP para los comandos, y la persistencia de la conexión WS para la transmisión continua.

5.1.2 Capas y módulos del backend

El backend está implementado íntegramente en *Python* 3.14 sobre el *framework FastAPI*. Se utiliza la versión *free-threaded* de Python (3.14t, PEP 703) [Gro23] para aprovechar el paralelismo de la Central Processing Unit (CPU) en el cómputo de física. El código se organiza en cinco capas con dependencias unidireccionales hacia abajo, como se muestra en la Figura 5.2. Cada capa expone su funcionalidad mediante interfaces explícitas y depende únicamente de las inmediatamente inferiores; el grafo de importaciones es acíclico, lo que facilita las pruebas y el despliegue parcial.

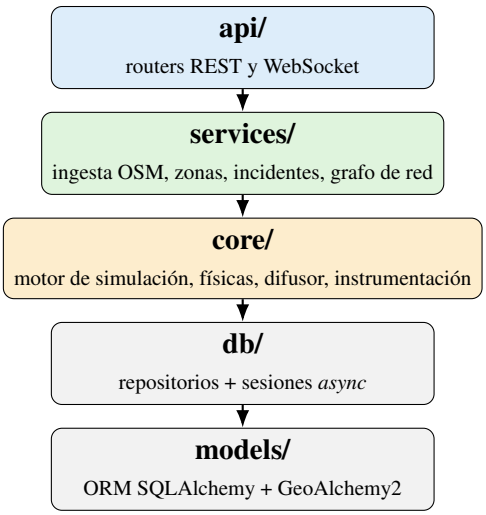

Figura 5.2: Capas del backend y dirección permitida de las dependencias entre paquetes

Sobre esta separación se apoyan dos patrones clásicos [Fow02]: el patrón *Repository*, que encapsula el acceso a la base de datos tras un contrato genérico, y la inyección de dependencias de *FastAPI*, que suministra a cada *endpoint* la sesión de base de datos y los servicios que necesita. Así cada *endpoint* es una función de sus dependencias y las pruebas pueden sustituirlas por dobles. Al arrancar, la aplicación construye el grafo en memoria (§ 5.2.4) e importa automáticamente el mapa OSM si la base de datos está vacía, de modo que el sistema esté listo en cuanto acepta peticiones.

5.1.3 Capas y módulos del cliente Godot

El cliente está implementado en GDScript sobre *Godot* 4.6 y persigue la misma estratificación que el backend, aunque la expresa con los medios propios del motor. La aplicación no se organiza como un grafo de importaciones entre paquetes, sino como un árbol de escenas cuyos nodos se comunican mediante *señales y ranuras* (*signals-slots*).

Sobre ese modelo se distinguen dos niveles. El primero es la capa de servicios, formada por *autoloads*: nodos globales que el motor instancia antes que cualquier escena y que hacen de *instancias únicas* (*singletons*) [GHJV94]. Cada uno gobierna un dominio de la aplicación, desde la comunicación con el backend hasta cada familia de entidades simuladas (vehículos, semáforos, incidentes, zonas, métricas), y anuncia sus cambios mediante señales, sin conocer

a sus receptores. El segundo nivel es la capa de presentación, que se suscribe a esas señales: los renderizadores de las entidades, la interfaz del operador y la cámara, apoyados en una representación propia de la red vial y en utilidades comunes. Su funcionamiento se describe en la Iteración 5 (§ 5.5.4).

Frente a la alternativa de inyectar las dependencias escena a escena, los *autoloads* son la forma idiomática en *Godot* de evitar que todas las escenas se acoplen a la jerarquía de la principal. Su coste, asumido conscientemente, es perder la inyección por constructor que sí practica el backend.

5.1.4 Despliegue

Sobre la separación cliente-servidor descrita, la unidad de despliegue del backend es un `docker-compose.yml` con tres servicios: el contenedor *postgis/postgis:15-3.3* que aloja la base de datos, el contenedor *backend* construido a partir del `Dockerfile` del proyecto y un *pgAdmin* de apoyo para inspección manual. Los volúmenes y la red privada `dt-network` aíslan al servicio del entorno del operador; el cliente Godot es un ejecutable nativo independiente del contenedor de servicios, que solo necesita conectividad HTTP/WS con el backend.

El orquestador arranca el contenedor del backend solo cuando la base de datos está lista —mediante una comprobación de salud— y monta el código como volúmenes, lo que permite desarrollar en caliente sin reconstruir la imagen.

5.2 Iteración 2: Datos y persistencia

La segunda iteración dotó al sistema de su materia prima: la red vial real sobre la que simular. El incremento desarrollado consistió en un flujo de procesamiento completo que importa la cartografía de Talavera desde OSM, la persiste en una base de datos geoespacial y construye con ella el grafo que el simulador recorre. A partir de un único fichero OSM, la red queda lista en memoria y en disco.

5.2.1 Modelo de datos relacional-espacial

El recorrido por el sistema empieza por su cimiento: los datos. Antes de simular nada hay que representar fielmente la red vial y almacenarla de forma que pueda consultarse con eficiencia, que es el segundo objetivo parcial. El modelo físico se recoge en la Figura 5.3. Todas las geometrías se almacenan en el sistema de referencia WGS84 [Ope10, OH21].

La elección de PostgreSQL con la extensión PostGIS frente a otras alternativas se justifica por tres factores: (a) madurez de sus operadores espaciales (`ST_Intersects`, `ST_Distance`, etc.), usados en la consulta de zonas, la serialización de geometrías y la búsqueda del nodo más próximo; (b) los índices GiST sobre R-tree que, como se detalló en la § 3.3.3, llevan a coste logarítmico la consulta `ST_Intersects` de una arista frente al polígono de una ZBE; (c) extensibilidad de la propia base de datos hacia futuras integraciones sin migrar a otro

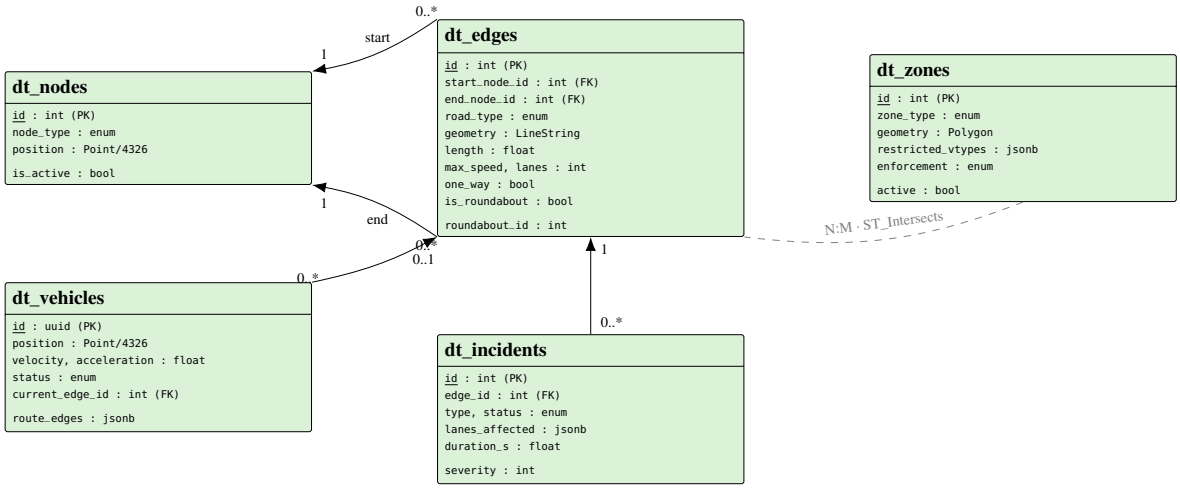

Figura 5.3: Esquema relacional de la base de datos. Cada tabla muestra sus columnas principales con su tipo y la marca de clave primaria (PK) o foránea (FK); todas las geometrías usan Spatial Reference System Identifier (SRID) 4326 (WGS84). Las aristas referencian a los nodos por dos claves foráneas (`start_node_id`, `end_node_id`) con borrado en cascada, y tanto incidentes como vehículos referencian a la arista en la que ocurren o circulan. La relación zonas–aristas no es una clave foránea, sino una correspondencia N:M derivada en consulta mediante `ST_Intersects`

motor.

Tres columnas del modelo guardan datos como `jsonb` en lugar de normalizarlos en tablas puente. La decisión responde a su patrón de acceso: estos campos se leen y escriben *siempre en bloque* y nunca se consultan con *join* ni se filtran desde SQL, porque el cálculo de rutas y la física operan sobre el grafo en memoria, no sobre la base de datos (§ 5.2.4). El *trade-off* —se renuncia a la integridad referencial y a la consulta indexada de esos campos— es asumible precisamente porque ninguna consulta del sistema los necesita en ese modo.

Cada tabla incorpora índices sobre los atributos que se consultan con frecuencia: índices espaciales GiST sobre las columnas geométricas (para las consultas de proximidad y de intersección con el polígono de una zona) e índices B-tree sobre los atributos de filtrado y sobre las claves que enlazan aristas con nodos, de modo que el grafo en memoria pueda reconstruirse sin recorrer las tablas completas.

El modelo emplea diversos atributos enumerados (tipo de nodo, clase de vía, estado del vehículo o tipo de incidente). En cuanto a la validación de los datos, la integridad referencial se define directamente en el esquema de la base de datos mediante claves foráneas con borrado en cascada (Figura 5.3). Sin embargo, el resto de restricciones —como la obligatoriedad, los rangos o la validación de los enumerados— se delega deliberadamente a la capa de aplicación a través de los modelos *Pydantic* de la API (§,5.5.1) y del ORM. Esta separación mantiene el esquema portable y concentra las reglas en la lógica de negocio. Por último, cabe destacar el atributo `length`, el cual es *derivado*: se calcula a partir de la geometría de la vía durante su ingesta y permanece estable mientras dicha geometría no cambie.

5.2.2 Migraciones Alembic

El esquema se gestiona como código mediante *Alembic*, en cuatro revisiones acumulativas. Se renuncia deliberadamente a la auto-generación de migraciones a cambio de un control exacto del Data Definition Language (DDL) emitido, ya que la auto-generación convive mal con los tipos de PostGIS y tiende a sugerir borrados espurios.

5.2.3 Proceso ETL desde OpenStreetMap

La red vial procede de archivos *.osm* (el formato Extensible Markup Language (XML) de OSM) descargados con *Overpass Turbo* para el área de Talavera. Sobre ellos se aplica un proceso ETL en cinco fases:

1. **Parseo** del documento XML, que recupera nodos y vías sin resolver aún sus referencias.
2. **Filtrado**: se conservan solo las vías transitables por vehículos (autovías, arterias, calles y sus enlaces) y se descartan aceras, carriles bici y zonas peatonales.
3. **Cálculo de cortes**: una vía de OSM representa una calle entera, pero en el simulador una arista debe terminar en cada intersección o semáforo. Se marcan como cortes los nodos compartidos por varias vías y los que tienen semáforo. Después, cada vía se segmenta entre cortes consecutivos. Las señales de *stop* y *ceda el paso*, situadas casi siempre sobre intersecciones, se reconocen en esta misma fase.
4. **Construcción de aristas**: para cada segmento se infieren el número de carriles, la velocidad máxima (con el valor por defecto del tipo de vía si falta) y el sentido, y se construye su geometría.
5. **Inserción por lotes** en la base de datos, para no penalizar el rendimiento con una orden por arista.

La red completa de Talavera se importa en unos pocos segundos a partir de un fichero de unos $15\,\text{MB}$: de los $\sim 58\,000$ nodos brutos del fichero OSM, el filtrado de vías transitables retiene unos $5\,900$ (en su mayoría vértices de geometría); la segmentación produce unas $2\,500$ aristas no dirigidas —$3\,400$ tras desdoblar las calles de doble sentido (§ 5.3.5)—, cuyos extremos son los $\sim 1\,800$ nodos topológicamente relevantes (intersecciones, semáforos y extremos de vía) que articulan un grafo conexo sobre el que se calculan las rutas.

Adicionalmente, las rotondas se reconocen en una pasada posterior que agrupa las aristas de cada anillo bajo un identificador común y las marca como tales. Esa información alimenta el tránsito por rotondas (§ 5.3.4, con *splines Catmull-Rom*) y el recorte de geometría del cliente Godot.

Este proceso no contiene nada específico de Talavera: opera sobre cualquier fichero OSM de entrada. Basta sustituir ese fichero (descargable de *Overpass Turbo* para cualquier área del mundo) para reconstruir y simular la red de otro municipio, materializando así el *condicio-*

nante «reentrenable a otra ciudad» del Cuadro 5.1. Esta independencia del municipio se ha comprobado importando sin cambios la red de Toledo.

5.2.4 Grafo en memoria sobre NetworkX

El simulador no consulta la base de datos en cada *tick*: tras la importación se construye una sola vez en memoria un grafo dirigido con *NetworkX* [HSS08], una biblioteca de referencia para el manejo de grafos en Python. Cada nodo conserva su posición y su tipo, y cada arista, los atributos que la física necesita —número de carriles, velocidad máxima, longitud, tipo de vía, geometría densa y su condición de rotonda— junto con un peso por defecto w derivado del tiempo de viaje libre. Mantener esta copia en memoria evita un acceso a disco en la ruta crítica, a cambio de asumir el compromiso de que toda mutación de la red se propague también al grafo.

El peso de cada arista combina dos factores:

$$w(u,v) = \frac{\ell(u,v)}{v_{\text{máx}}(u,v)} \cdot \rho(\text{road\_type}) \tag{5.1}$$

donde ℓ es la longitud en metros, $v_{\text{máx}}$ la velocidad máxima de la vía en m/s y ρ un factor de penalización por tipo de vía. El factor ρ **no se calibra con datos de tráfico**: es una *constante de diseño* fijada por tipo de vía con un criterio simple: premiar las vías de mayor jerarquía y penalizar las de menor. Toma valores entre $0{,}6$ para una autovía y $1{,}8$ para una calle residencial, de modo que desincentiva desviarse a una residencial para ahorrar unos metros cuando existe una arteria principal. El valor mínimo de ρ se reutiliza como cota inferior en la heurística de A* (§ 5.3.5).

Sobre el grafo se construyen además índices auxiliares para las rotondas (las aristas que forman cada anillo y su longitud acumulada), útiles para el arbitraje de prioridad (§ 5.4.3). El grafo expone también una capa de *pesos dinámicos* que el motor de simulación actualiza sobre la marcha en función de la saturación de las rotondas, sin reconstruirlo. Las restricciones de zonas e incidentes, en cambio, no pasan por esa capa: se inyectan en A* como conjuntos de aristas penalizadas (§ 5.3.5). En todos los casos, cualquier mutación que afecte al grafo se propaga de forma síncrona, de modo que las rutas se calculan siempre sobre el estado vigente de la red.

5.3 Iteración 3: Núcleo de simulación

Con la red vial ya persistida y disponible como grafo en memoria (§ 5.2), el sistema dispone del *escenario* sobre el que circular. Falta poblarlo de vehículos que se muevan de forma realista por él. De eso trató la tercera iteración, cuyo incremento fue el primer tráfico vivo recorriendo el mapa. Responde al tercer objetivo parcial: *desarrollar el modelo de comportamiento vehicular, gestor de vehículos y sistema de cálculo de rutas*. Se ha optado

por un modelo microscópico continuo en el espacio y discreto en el tiempo, en línea con la tradición fundada por [Gip81] y consolidada en [TK13]. El núcleo se compone de IDM para el seguimiento longitudinal, MOBIL para los cambios de carril y *splines Catmull-Rom* centrípetos para el tránsito por geometrías curvas, sobre una malla de rutas calculadas con A*.

Antes de entrar en cada pieza se fija el *modelo de dominio* [Eva03] del *runtime*, complementario al modelo físico de la base de datos (Figura 5.3): el motor de simulación coordina los vehículos y las entidades de control (semáforos, incidentes y zonas, § 5.4), y cada vehículo sigue una ruta sobre el grafo de aristas y nodos. Cada entidad mantiene hasta *tres* representaciones —fila en la base de datos, objeto en memoria e instancia renderizada en el cliente—, de las cuales el objeto en memoria es la *fuente de verdad* en ejecución. Toda mutación de la red se propaga de forma síncrona al grafo (§ 5.2.4).

5.3.1 Tipos de vehículo

El motor distingue tres clases de vehículo con perfiles físicos diferenciados (Cuadro 5.2). La distinción no es meramente estética: actúa directamente sobre los parámetros físicos y, por tanto, sobre el comportamiento emergente. Un camión utiliza $T = 2{,}4\,\text{s}$ y una deceleración cómoda de $b = 2{,}0\,\text{m/s}^2$, lo que reproduce la característica cola que se forma tras él en una arteria; una motocicleta admite un espaciado mínimo reducido ($s_0 = 1{,}0\,\text{m}$) y se incorpora a huecos en los que un turismo no entra.

Tipo	**Long.** (m)	**Anch.** (m)	v_0 (m/s)	s_0 (m)	T (s)	a (m/s²)	b (m/s²)
Turismo (*car*)	4,5	1,9	13,89	2,0	1,4	1,4	2,5
Motocicleta (*moto*)	2,1	0,8	15,28	1,0	1,0	2,0	3,5
Camión (*truck*)	10,0	2,5	11,11	3,0	2,4	0,7	2,0

Cuadro 5.2: Parámetros físicos y de comportamiento por tipo, expresados en unidades del IDM: dimensiones espaciales (Long., Anch.), velocidad deseada en vía libre (v_0, equivalentes a 50, 55 y $40\,\text{km/h}$ respectivamente), distancia de parada mínima (s_0), margen temporal de seguridad (T), aceleración máxima (a) y deceleración cómoda (b)

La Figura 5.4 muestra, con un diagrama de clases, cómo se organizan estos tipos en el código. Cada tipo se resume en un *perfil* (la clase `VehicleTypeProfile`) que reúne todos sus parámetros: dimensiones, velocidad máxima y los valores del IDM. Los tres perfiles se guardan juntos en un único catálogo (`PROFILES`), del que parte todo lo demás. Cuando se crea un vehículo, recibe un tipo al azar (con más probabilidad un turismo que una moto o un camión) y guarda solo una referencia a él, sin copiar sus parámetros, que viven una única vez en el perfil. En cada *tick*, el motor consulta ese perfil para aplicar a cada vehículo la física que le corresponde.

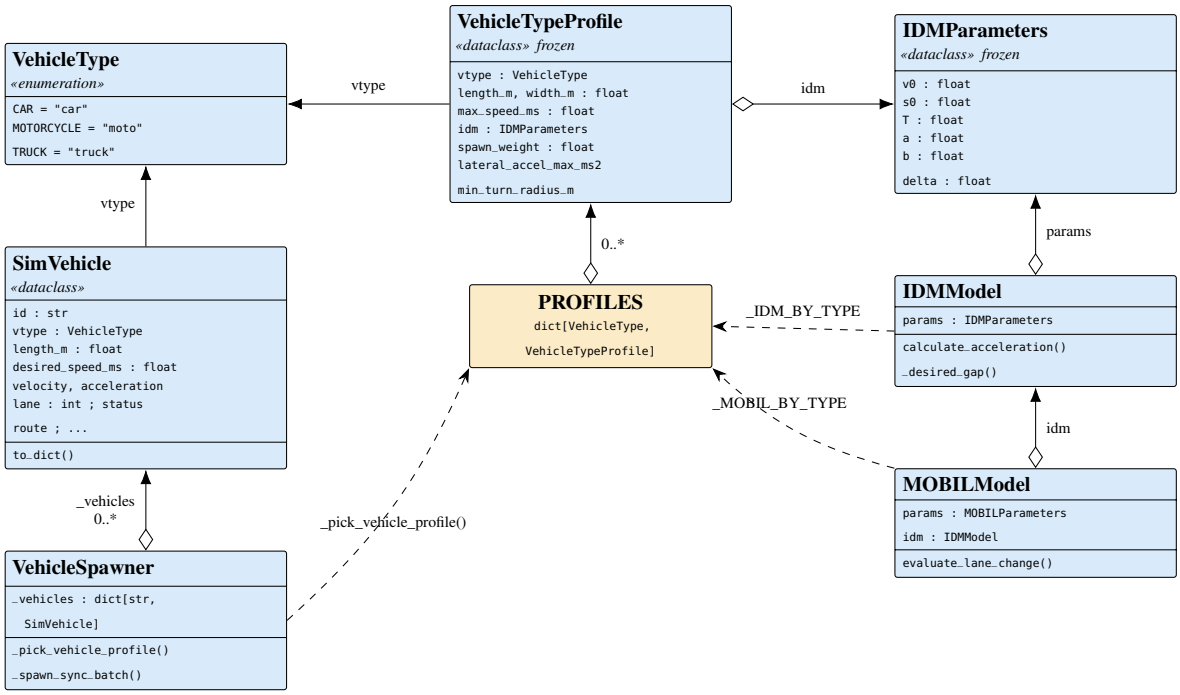

Figura 5.4: Diagrama de clases UML del subsistema de tipos de vehículo. Cada `VehicleTypeProfile` reúne los parámetros de un tipo (incluido su bloque `IDMParameters`) y todos se registran en el catálogo `PROFILES`, del que dependen el generador (`VehicleSpawner`) y los modelos de física por tipo (`IDMModel` y `MOBILModel`). El generador crea los vehículos en memoria (`SimVehicle`), que solo guardan una referencia a su tipo.

Para incorporar un nuevo tipo de vehículo en el futuro, como por ejemplo un autobús, basta con definir su perfil (dimensiones, parámetros y probabilidad de aparición) y registrarlo en el catálogo. El motor lo recoge sin ningún otro cambio y el nuevo tipo empieza a circular con su física propia.

5.3.2 Intelligent Driver Model

El IDM [THH00] modela la aceleración longitudinal de un vehículo a partir de su velocidad, su velocidad deseada y la distancia y velocidad relativa respecto al de delante; su formulación y la interpretación de cada parámetro se introdujeron en el Capítulo 3. Se adopta tal cual, con el exponente estándar $\delta = 4$ y los parámetros diferenciados por tipo de vehículo (Cuadro 5.2). El caso de *carretera libre* (sin vehículo delante) se trata aplicando solo el término de velocidad libre.

Sobre el modelo base, el motor introduce *líderes virtuales*: un semáforo en rojo se materializa como un vehículo fantasma de velocidad nula situado a la distancia de la línea de *stop*; un cruce no señalizado en el que el vehículo debe ceder el paso, como un líder a la distancia de incorporación con la velocidad del vehículo cruzante. Esta abstracción permite reaprovechar el IDM para frenar ante eventos no longitudinales, sin lógica adicional.

5.3.3 MOBIL: cambios de carril

El IDM resuelve la dinámica longitudinal pero no la lateral. Para los cambios de carril se emplea MOBIL [KTH07], introducido también en el Capítulo 3: decide el cambio comparando las aceleraciones IDM de los vehículos afectados antes y después de un hipotético cambio, bajo un criterio de seguridad y otro de incentivo. Se adoptan un frenazo máximo tolerable $b_{\text{safe}} = 4$ m/s^2, un factor de cortesía $p = 0{,}5$ (conductor moderadamente altruista) y un umbral de incentivo $\Delta a_{\text{th}} = 0{,}2$ m/s^2, por debajo del cual no compensa cambiar de carril.

La implementación evalúa hasta dos candidatos por *tick* —el carril izquierdo y el derecho— y devuelve el de mayor incentivo. Sobre la formulación clásica se han añadido dos detalles operacionales:

- **Distancia de seguridad mínima**: se rechazan de entrada los candidatos cuyo hueco libre por delante o por detrás sea inferior a 1 m, antes incluso de aplicar la fórmula IDM, lo que evita que huecos muy pequeños o negativos produzcan evaluaciones de seguridad sin sentido físico.
- **Cambio forzado**: si un vehículo queda atrapado en un carril cerrado por un incidente, el motor fuerza el cambio y MOBIL acepta el mejor candidato seguro aunque su incentivo sea negativo, garantizando que el vehículo desaloje el carril. Esta extensión es la que acopla MOBIL con el cierre dinámico de carriles (§ 5.4.4).

5.3.4 Movimiento por puntos de paso y splines

Los modelos IDM/MOBIL suministran aceleración y carril, pero no posición geográfica. La traducción de aceleración a coordenadas (λ, φ) exige una representación geométrica de la arista. La forma directa —usar el *LINESTRING* de PostGIS y avanzar entre vértices— produce comportamientos antinaturales en rotondas, donde los vértices originales de OSM suelen ser pocos (4-8 puntos para describir un anillo entero) y un vehículo virtual «pivota» bruscamente cada vez que cambia de tramo.

La solución es transformar esas polilíneas como *splines Catmull-Rom centrípetos* [CR74, YSK11]: una curva suave que pasa por todos los puntos de control y que, en su variante centrípeta, evita los picos y rizos que la variante uniforme produce cuando los puntos están muy juntos, como en los anillos de las rotondas [YSK11]. Las polilíneas se simplifican antes con el algoritmo Ramer-Douglas-Peucker (RDP) [Ram72, DP73], y la curva resultante —junto con su tabla de longitudes acumuladas— se precalcula al construir el grafo, de modo que en tiempo de ejecución la posición de un vehículo a lo largo de la arista se obtiene de forma eficiente.

Esta misma aritmética debe coincidir en el servidor (que calcula las posiciones) y en el cliente (que las interpola entre *ticks*, § 5.5.4): si divergieran, el vehículo «saltaría» en cada actualización. Por eso el cliente reproduce el mismo algoritmo con idénticas constantes, y la implementación de referencia del servidor se valida con pruebas automáticas.

5.3.5 Cálculo de rutas con A*

Cada vehículo lleva una ruta como secuencia de aristas. El cálculo se delega a A* [HNR68] sobre el grafo `NetworkX` descrito en § 5.2.4. La heurística usada es una cota inferior de tiempo de viaje basada en la distancia geográfica:

$$h(u) \;=\; \frac{d_{\text{euc}}(u, \text{end}) \cdot \rho_{\text{mín}}}{v_{\text{máx}}} \tag{5.2}$$

donde d_{euc} es una distancia euclídea *aproximada* calculada a partir de las coordenadas geográficas (λ, φ) de los nodos. Se obtiene multiplicando la distancia angular $\sqrt{\Delta\lambda^2 + \Delta\varphi^2}$ por la constante $111\,320$ m/grado (la longitud aproximada de un grado de arco sobre un meridiano terrestre). Los otros dos términos son *constantes*: $\rho_{\text{mín}}$ es el valor mínimo de la tabla de factores por tipo de vía de la § 5.2.4; y $v_{\text{máx}} = 130$ km/h $\approx 36{,}1$ m/s es el límite legal máximo de velocidad en autovía en España, tomado como cota superior de la velocidad alcanzable en la red. Emplear el factor de penalización *mínimo* y la velocidad *máxima* garantiza que la heurística nunca sobreestime el tiempo real y A* sigua devolviendo el camino óptimo.

A esa heurística se superpone una capa de *pesos dinámicos* que encarece ciertas aristas según el estado vivo de la simulación. Las aristas cortadas reciben un multiplicador prohibitivo ($\times 10^9$), que las hace en la práctica intransitables; y las que caen dentro de una zona restringida para el tipo de vehículo actual reciben uno más moderado ($\times 50$), que las desincentiva con fuerza pero sin descartarlas por completo. La diferencia de magnitud entre ambos factores es intencionada: un carril cortado por un accidente o evento debe evitarse siempre que sea físicamente posible, mientras que una ZBE admite que un vehículo restringido la atraviese si la única alternativa es un rodeo desproporcionado. La clave del diseño es que, en ninguno de los dos casos, las aristas se eliminan del grafo: solo se penalizan. Así, cuando no existe ninguna ruta alternativa, A* devuelve igualmente el camino penalizado y la simulación nunca se queda sin ruta, una garantía que se valida explícitamente en el Capítulo 6 (§ 6.3). Ambos factores responden a un razonamiento de órdenes de magnitud, no a una calibración con datos: lo determinante no es la cifra exacta, sino la separación de unos siete órdenes de magnitud entre ellos, que traduce a aritmética de costes la distinción entre una prohibición absoluta y un disuasorio firme pero permeable.

Las rutas calculadas se cachean por la tripla origen-destino-tipo de vehículo para no recalcular trayectos idénticos durante las generaciones masivas, y la caché se invalida al cambiar los pesos dinámicos o activar una zona, de modo que las rutas reflejan siempre el estado de la red.

5.4 Iteración 4: Control y eventos

El núcleo de la iteración anterior basta para que los vehículos circulen, pero un gemelo digital pensado para escenarios *what-if* necesita además la capacidad de *alterar* ese tráfico

sobre la marcha y observar cómo reacciona. Ese es justamente el cuarto objetivo parcial, que pide un *sistema de control y gestión de eventos*: gestor de semáforos, gestor de incidentes, cierre/apertura dinámico de carriles y gestor de zonas esenciales. Esta sección describe cómo se implementan, coordinados desde el motor central de simulación, que se presenta primero.

5.4.1 Motor de simulación con tick fijo

El bucle principal sigue una máquina de estados finita (Finite State Machine (FSM)) con cuatro estados (Figura 5.5). Las transiciones se inician desde la API REST (§ 5.5), que rechaza con un código HTTP 409 las que no son válidas en el estado actual.

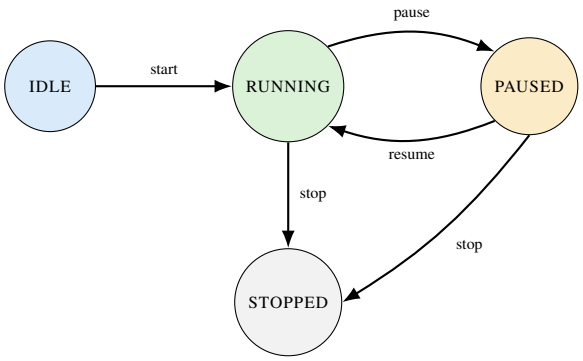

Figura 5.5: Diagrama de estados Unified Modeling Language (UML) del motor de simulación. Las transiciones *start*, *pause*, *resume* y *stop* las dispara la API REST

El bucle se ejecuta dentro del bucle de eventos del servidor a una frecuencia configurable (3 Hz por defecto). En cada *tick*:

1. avanza los temporizadores de semáforos e incidentes;
2. calcula la física de los vehículos (IDM, MOBIL, arbitraje de cruces e integración temporal);
3. construye la instantánea de estado y la difunde a los clientes.

Una sutileza no trivial es el desacoplamiento entre cómputo y difusión. Si el motor esperara a que la difusión del *tick* N llegara a todos los clientes WS antes de empezar el *tick* $N+1$, los clientes lentos atrasarían el cómputo. Para evitarlo, el motor lanza la difusión como una tarea no bloqueante y, al iniciar el siguiente *tick*, solo espera por ella si aún no ha terminado. Así, física y red corren en paralelo dentro del mismo bucle de eventos y un cliente lento apenas afecta al ritmo del cómputo.

La configuración (frecuencia de *tick*, parámetros IDM/MOBIL) puede actualizarse en caliente desde la API sin reiniciar la simulación, lo que resulta útil para afinar el comportamiento mientras corre.

El motor no *contiene* la lógica de control, sino que *coordina* un conjunto de colaboradores especializados (el generador de vehículos, el difusor de estado, el controlador de semáforos

y los gestores de incidentes y zonas), que se describen en las subsecciones siguientes y comparten el grafo de la red.

5.4.2 Controlador de semáforos

El controlador de semáforos es deliberadamente simple: implementa ciclos fijos de tiempo, sin sensores adaptativos. Un ciclo fijo basta para validar el resto del sistema (cola tras rojo, descarga tras verde). El ciclo es verde $30\,\mathrm{s} \rightarrow$ ámbar $5\,\mathrm{s} \rightarrow$ rojo $35\,\mathrm{s}$ (periodo de $70\,\mathrm{s}$), y cada semáforo arranca con una fase inicial aleatoria para evitar que toda la malla se sincronice de forma artificial.

Las aristas que entran en una intersección se reparten en dos grupos según su orientación (los dos sentidos de un mismo eje van juntos) y cada grupo recibe la fase opuesta, de modo que un eje está en verde mientras el perpendicular está en rojo.

El controlador ofrece dos modos de anulación manual (*override*), por nodo y global, pensados para escenarios *what-if* y para que el operador pueda actuar desde el Heads-Up Display (HUD) de Godot (§ 5.5.4).

5.4.3 Cruces no señalizados, STOP, ceda el paso y rotondas

Talavera tiene muchos más cruces no señalizados que con semáforo. La regla por defecto es *prioridad a la derecha*, modulada por las señales de `stop` y `give_way` (ceda el paso) detectadas durante la importación OSM (§ 5.2.3).

El arbitraje se proyecta como un *líder virtual* para el IDM: si un vehículo no tiene prioridad, ve un líder a la distancia de la línea de *stop* con velocidad cero, lo que produce una desaceleración natural. Las dos sutilezas operativas son:

- **STOP con tiempo de detención efectiva**. Una señal de *stop* no es solo una desaceleración hasta cero: el vehículo debe *detenerse durante un tiempo* antes de reanudar la marcha. Para modelarlo, cuando el vehículo está a menos de $2\,\mathrm{m}$ de la señal y casi parado (velocidad inferior a $0{,}5\,\mathrm{m/s}$), un contador acumula el tiempo detenido; al superar el segundo de detención, la señal se da por atendida y el vehículo recupera su comportamiento IDM libre.

- **Rotondas**. El vehículo que ya circula por el anillo tiene prioridad y el que pretende entrar cede. Sobre esa regla, el motor aplica la *aceptación de huecos* de la § 3.1.3, evaluada solo a menos de $30\,\mathrm{m}$ de la línea de entrada. Para cada vehículo del anillo en el carril de destino se estima el tiempo hasta el punto de inserción (su distancia de arco, precalculada al construir el grafo, entre su velocidad actual), y el entrante cede si ese tiempo es inferior a $3\,\mathrm{s}$ o el hueco menor de $8\,\mathrm{m}$. La doble condición —tiempo *y* distancia mínima— evita tanto las incorporaciones temerarias a alta velocidad como las que se cuelan en un hueco demasiado corto a baja velocidad.

En los cruces perfectamente simétricos (varios vehículos llegando a la vez) en los que la prioridad a la derecha no llega a decidir, un desempate determinista por identificador rompe la simetría de forma reproducible y evita que el cruce se quede bloqueado.

5.4.4 Gestor de incidentes y cierre dinámico de carriles

Un *incidente* es una interrupción puntual sobre una arista, con un tipo (accidente, obra, avería…), una severidad y unos carriles afectados. El gestor de incidentes administra su ciclo de vida (alta, expiración automática, retirada y extensión): al darse de alta uno, lo persiste en base de datos, lo proyecta al estado vivo de la simulación, reenruta a los vehículos directamente afectados y difunde el evento WS a los clientes.

La proyección al estado vivo es lo que da fuerza al modelo: un incidente se traduce en dos efectos que la física consulta en cada *tick*:

Aristas bloqueadas. Las aristas cortadas íntegramente. A* las penaliza —no las elimina (§ 5.3.5)— y el cálculo de rutas las evita siempre que puede.

Carriles cerrados. Los carriles cerrados de cada arista. MOBIL descarta los candidatos que apunten a ellos y, para los vehículos atrapados en uno, fuerza el cambio de carril (§ 5.3.3).

La detección de colisiones cierra el bucle: cuando la física detecta un choque, se registra como un incidente de tipo accidente que alimenta esas mismas estructuras. El registro es inmediato y no espera a la base de datos (la escritura se difiere fuera de la ruta crítica), lo que mantiene la física desacoplada de la persistencia y preserva la coherencia aunque la base de datos no esté disponible.

Los incidentes se retransmiten al cliente al darse de alta, modificarse o retirarse, y este los representa como cilindros 3D sobre la arista afectada, con color según su severidad (§ 5.5.4).

5.4.5 Gestor de zonas: implementación de la ZBE

Las zonas se modelan como polígonos asociados a una política de control. Al darse de alta una zona, se precalcula qué aristas caen dentro de ella y se guarda en memoria, de modo que el cálculo de rutas y la generación de vehículos lo consultan sin coste apreciable.

Los tres modos de aplicación de la restricción (*enforcement*) son intencionalmente distintos en sus consecuencias:

- `warn`: la zona se renderiza pero no afecta al cálculo de rutas. Usado para preavisar de un perímetro futuro.
- `deny_spawn`: el generador de vehículos no crea vehículos de los tipos restringidos cuyo origen o destino caiga dentro de la zona. Es la política más estricta y modela el caso límite de una ZBE bien implantada.

- **`force_reroute`**: las aristas internas reciben en A* un multiplicador de penalización ($\times 50$) para los tipos de vehículo restringidos, como se describió en la § 5.3.5. Al penalizar sin prohibir, los vehículos pueden seguir entrando cuando la alternativa es desproporcionadamente larga, lo que reproduce mejor la ZBE «no sancionadora» aprobada por Talavera (§ 3.5.3).

En los tres modos, la discriminación se realiza por *tipo de vehículo* (`car`, `moto`, `truck`), que es la unidad de restricción que maneja el prototipo.

La caché de zonas mantiene, para cada tipo de vehículo, el conjunto de aristas que A* debe penalizar. Cuando una zona cambia, ese conjunto se recalcula entero y se reemplaza de una sola vez, de modo que las lecturas de la física ven siempre una versión coherente —la anterior o la nueva, nunca una a medio actualizar— sin necesidad de un *lock* que las serialice [Bea10]. Además, al crear o modificar una zona solo se reenruta de inmediato a unos pocos centenares de vehículos afectados y el resto se reencamina en una tarea en segundo plano: la cota evita que la petición HTTP del operador se bloquee cuando la zona afecta a miles de rutas activas, a cambio de que la corrección del conjunto sea eventual en lugar de instantánea.

5.5 Iteración 5: Cliente 3D y comunicación

Las cuatro iteraciones anteriores construyeron un simulador completo pero sin interfaz con el exterior. Esta quinta iteración produjo la capa que lo convierte en una herramienta usable, cubriendo los objetivos parciales quinto y sexto: por el lado del servidor, una API REST para los comandos del operador (§ 5.5.1), un canal WS que emite el estado en tiempo real (§ 5.5.2) y un motor de analíticas con las métricas clave del tráfico (§ 5.5.3); por el lado del cliente, una aplicación *Godot* que recibe ese flujo y lo convierte en una escena 3D interactiva (§ 5.5.4).

5.5.1 API REST con FastAPI

La API se monta sobre *FastAPI*, un *framework* web asíncrono de Python que valida automáticamente los datos de entrada y salida. Los *endpoints* se agrupan por dominio y reciben por inyección de dependencias la sesión de base de datos y los servicios que necesitan. En conjunto, la API cubre la importación del mapa, el control de la simulación (arranque, pausa, parada y configuración en caliente), la generación y el reenrutado de vehículos, la gestión de incidentes y zonas, y la consulta de métricas. Los errores de dominio se traducen a los códigos HTTP adecuados: pausar una simulación detenida devuelve `409 Conflict`, y un vehículo inexistente, `404 Not Found`.

La idempotencia se ajusta con precisión a la semántica HTTP: las consultas (`GET`) y las sustituciones o bajas (`PUT`, `DELETE`) son idempotentes, mientras que las creaciones (`POST` de vehículos, incidentes o zonas) *no* lo son, pues cada invocación produce un recurso nuevo. Las peticiones se validan de forma declarativa y los esquemas completos de petición y respuesta, junto con la documentación interactiva de la API, se autogeneran en `/docs`.

5.5.2 WebSocket: protocolo y difusión

La transmisión de estado se realiza sobre un único punto de entrada WS. Un gestor de conexiones mantiene la lista de clientes activos y les envía cada mensaje de forma concurrente, de modo que un cliente lento no retrasa al resto.

El protocolo cubre nueve tipos de mensaje: el *tick* de estado (cada 333 ms, en el formato binario de la § 5.6.1), los cambios de estado de la simulación, las altas y bajas de vehículos, las colisiones, las fases de los semáforos, los incidentes, las zonas y el cambio de mapa. Salvo el *tick*, que es binario por su volumen, el resto viaja en JSON [Bra17], cómodo de depurar. Todos van en dirección *servidor* $\rightarrow$ *cliente*. La entrega y el orden se apoyan en las garantías de Transmission Control Protocol (TCP) sobre las que opera WS; en cada *tick*, el difusor serializa el estado de los vehículos activos y, si el mensaje resulta muy grande, lo divide en fragmentos numerados que el cliente solo aplica al completo (§ 5.5.4). La Figura 5.6 reconstruye la coordinación de extremo a extremo dentro del servidor asíncrono: el cómputo de la física del *tick* en el grupo de hilos, su serialización y difusión binaria concurrente, y la propagación de los eventos de dominio en JSON hacia los clientes Godot.

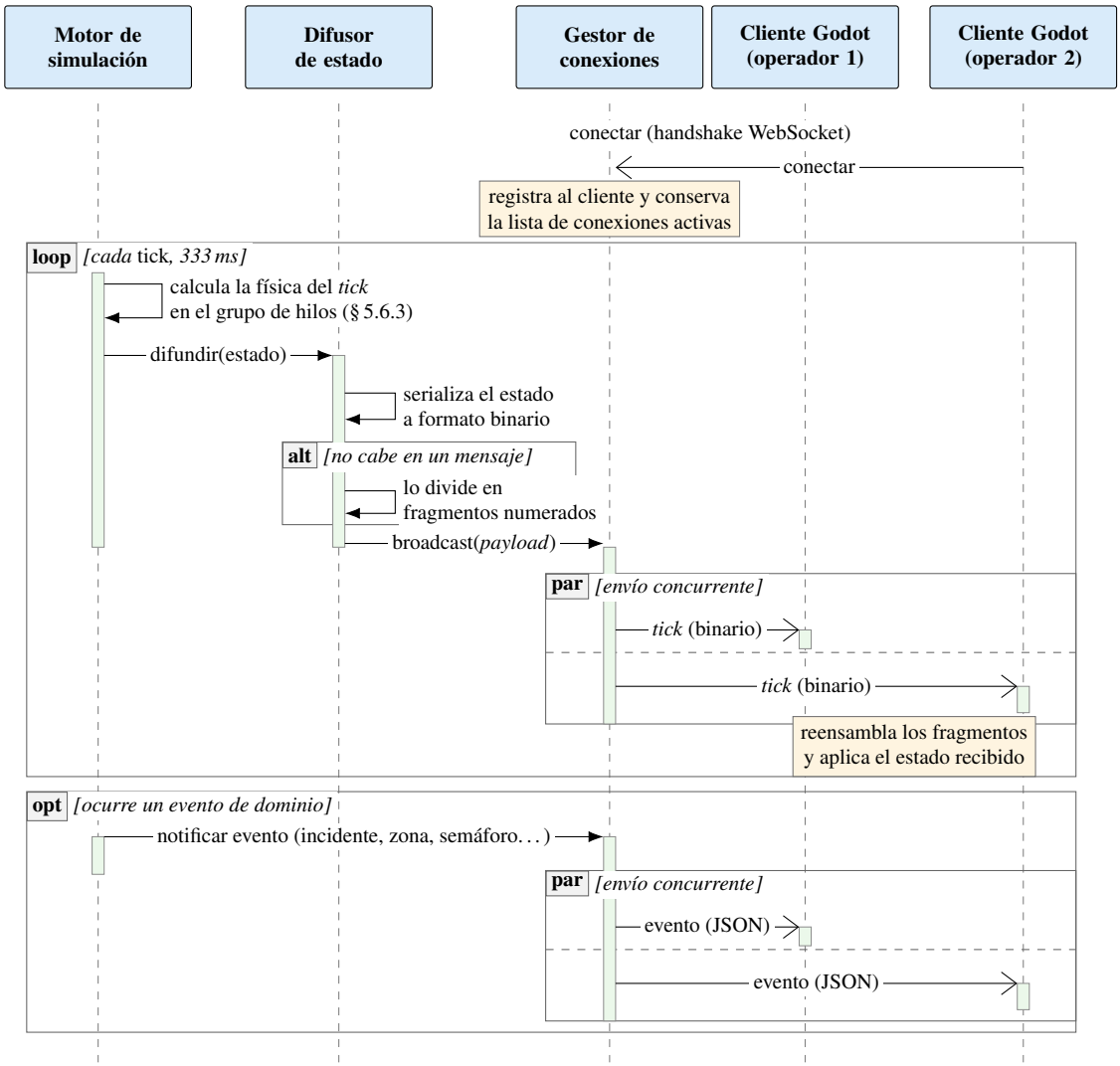

Figura 5.6: Diagrama de secuencia UML de la difusión de estado. Tras la conexión inicial, el gestor de conexiones mantiene la lista de clientes. En cada *tick*, el motor calcula primero la física en el grupo de hilos (§ 5.6.3); el difusor serializa después el estado a binario y, si excede el tamaño máximo de un mensaje WS, lo divide en fragmentos numerados (fragmento *alt*); el envío a los clientes es concurrente (fragmento *par*), de modo que un cliente lento no retrasa al resto. Los eventos de dominio (incidentes, zonas, fases de semáforo) se propagan de forma asíncrona en JSON cuando se producen (fragmento *opt*). Las flechas de cabeza maciza denotan llamadas síncronas internas del servidor; las de cabeza abierta, mensajes asíncronos sobre la red

5.5.3 Motor de analíticas de tráfico

Aunque este componente se desarrolló dentro de la iteración del cliente —y por trazabilidad histórica se describe aquí—, es código del *servidor*. El *motor de analíticas* calcula las métricas de tráfico que un gestor municipal querría leer de un escenario y computa tres familias, elegidas por poder derivarse del estado de la simulación sin datos externos:

Tiempo de viaje. Al completar su ruta, cada vehículo aporta su tiempo de viaje y su velocidad media; sobre los viajes completados se reportan media, mediana, percentil 95 y máximo.

Nivel de congestión. Periódicamente se cuenta cuántos vehículos circulan por cada arista en relación con su *capacidad estimada* —definida como su longitud por su número de carriles dividida entre el espacio que un vehículo detenido ocupa en cola, fijado en 7 m por vehículo y carril—, y se agregan la media, el máximo y el número de aristas saturadas de esa razón ocupación/capacidad.

Impacto de incidentes. Se cuenta cuántos vehículos activos tienen por delante una arista bloqueada por un incidente, una medida directa del tráfico forzado a recalcular ruta.

5.5.4 Renderizado 3D e interfaz de usuario

El flujo de estado que emite el servidor no es más que una sucesión de números. Convertirlo en una escena 3D que un operador no técnico pueda entender y manipular es la misión del cliente *Godot*, y constituye el sexto y último objetivo parcial: *desarrollar el módulo de renderizado de mapas que incluya sistema de cámaras, panel de control de usuario y sistema de visualización de incidentes y zonas especiales*. Los apartados siguientes recorren la cadena completa desde las coordenadas geográficas hasta el HUD que ve el operador.

Conversión de coordenadas GPS-Godot

El servidor produce posiciones en WGS84 (EPSG 4326), pero el motor 3D necesita coordenadas locales en metros. El cliente reutiliza la proyección equirrectangular de la § 3.3.3 (Ecuación (3.5)), con (λ, φ) la longitud y latitud del punto y (λ_0, φ_0) el centro de la red (en Talavera, $(-4{,}830,\ +39{,}960)$), y la instancia con una salvedad propia del motor 3D:

$$x = (\lambda - \lambda_0) \cdot 111\,320\,\cos(\varphi_0) \tag{5.3}$$

$$z = -(\varphi - \varphi_0) \cdot 111\,320 \tag{5.4}$$

$$y = h_{\text{elev}} \quad (\text{altura, por defecto } 0)$$

La novedad respecto a la Ecuación (3.5) es la **inversión del eje** z: *Godot* sigue la convención *front* = $-Z$, mientras que la latitud crece hacia el norte; sin esa inversión los vehículos se moverían en sentido contrario al esperado por la cámara. La tercera componente, y, recoge la elevación (nula por defecto).

Renderizado masivo con MultiMesh

El reto de rendimiento del cliente es dibujar una flota de varios miles de vehículos a 60 FPS sin saturar la Graphics Processing Unit (GPU) (§ 6.7). La dificultad no está tanto en el número de triángulos como en el de órdenes de dibujo: emitir una llamada (*draw call*) por vehículo saturaría la comunicación CPU-GPU mucho antes de agotar la capacidad de cálculo gráfico. Lo resuelve la *instanciación* mediante MULTIMESHINSTANCE3D [AMHH+18]: una única malla base se dibuja en una sola llamada acompañada de N matrices de transformación, una por instancia, de modo que el coste de coordinación deja de crecer con N.

Cada vehículo se compone de varias mallas instanciadas (cuerpo, techo, ruedas y luces de freno), y un sistema de ranuras preasignadas permite dar de alta y de baja vehículos en tiempo constante, sin reordenar el resto de la flota. Las dimensiones de cada tipo de vehículo se aplican como un factor de escala por instancia sobre una única malla base, lo que evita gestionar mallas separadas por tipo y mantiene una sola llamada de dibujo. La Figura 5.7 (§ 5.7) muestra el cliente renderizando una flota completa con esta técnica.

La misma técnica se aplica a los nodos de la red (intersecciones y rotondas) y a los semáforos. A esto se añade un nivel de detalle progresivo (Level Of Detail (LOD)): los nodos lejanos se dibujan más pequeños o se ocultan, para que la vista alejada no se sature y siga siendo legible.

Geometría de aristas y rotondas

Las aristas se dibujan como geometría procedural: se extruye lateralmente la polilínea de la calle con un ancho proporcional al número de carriles, y el color codifica el tipo de vía. Las calles de un solo sentido reciben flechas para indicar la dirección. Las rotondas requieren un tratamiento especial, recortando los brazos entrantes a una distancia segura del anillo para que no invadan su interior. Cada tipo de geometría se dibuja en una capa independiente, de modo que al cerrar un carril solo se redibuja la capa de incidentes y el resto se conserva.

Interpolación en el cliente y extrapolación por estima

El servidor publica un *tick* cada 333 ms y el cliente renderiza a 60 FPS; sin interpolación, los vehículos saltarían entre posiciones en cada actualización. Para evitarlo, el cliente aplica *interpolación con retardo* [Ber01]. Mantiene las dos últimas posiciones conocidas de cada vehículo e interpola entre ellas con un desfase deliberado de 200 ms, lo que produce un movimiento fluido. Si el siguiente *tick* se retrasa, en lugar de congelar al vehículo se extrapola su posición a velocidad constante (la *estima* o *dead reckoning* de la navegación), con un tope para evitar derivas; y si una posición nueva se aleja demasiado de la prevista (por una reaparición o un recálculo brusco de ruta), el cliente salta directamente a ella. Esta aritmética coincide con la del servidor (§ 5.3.4), de modo que el cliente solo difiere de él en la temporización del *renderizado*, nunca en el cálculo geométrico.

Cliente WebSocket multihilo y HUD

El cliente WS usa un patrón productor-consumidor para que el parseo de un *tick* grande no bloquee el render: el hilo principal solo extrae los paquetes del socket y los encola, un hilo dedicado los deserializa, y el hilo principal consume el resultado en cada *fotograma*. El cliente reensambla además los *ticks* que llegan fragmentados antes de aplicarlos, para no mostrar un estado parcial.

La cámara es de tipo *vuelo libre* (la orientación se controla con el ratón y el desplazamiento con el teclado) y ofrece un modo de seguimiento que la ancla a un vehículo seleccionado. El HUD muestra el tiempo de simulación, el número de vehículos y los FPS, con pestañas por dominio y un panel lateral para el elemento seleccionado. Sobre él se montan los *modos de operador* (crear un incidente, cerrar un carril, dibujar una zona o reasignar el destino de un vehículo), cada uno con realimentación visual antes de confirmar. Así, el operador puede reproducir en segundos un corte de calles o la entrada en vigor de una ZBE sin tocar la línea de comandos. Una capa de diagnóstico sobreimpresa (tecla F3, el PERFMONITOR) muestra los FPS, la latencia WS y la memoria; la Figura 5.7 muestra el HUD en operación.

5.6 Iteración 6: Rendimiento y optimización

Con el sistema ya completo de extremo a extremo, la última iteración se dedicó por entero a que escalara de unos cientos a varios miles de vehículos sin perder la estabilidad del *tick*, reescribiendo tres puntos críticos de la ruta crítica: el formato binario en que el estado viaja del servidor al cliente (§ 5.6.1), la instrumentación que mide dónde se consume el tiempo (§ 5.6.2) y la paralelización del cómputo de física (§ 5.6.3).

5.6.1 Formato binario de transmisión

El origen del formato propio está en el prototipo inicial, donde el estado de cada *tick* se enviaba como JSON, la opción más cómoda de programar y depurar. Pero el coste de serializar ese JSON crece con el número de vehículos y, con flotas grandes, llegaba a bloquear el bucle de eventos durante varios milisegundos y a degradar la frecuencia efectiva del *tick*. La representación textual del estado se había convertido en el cuello de botella del servidor. La solución fue sustituirla, solo para el *tick*, por un formato binario *ad hoc*, compacto y con una cabecera de control que porta el identificador de formato, la versión del protocolo y la información de fragmentación.

Las dos decisiones clave son: (a) los valores en coma flotante se serializan en precisión simple en lugar de doble, lo que basta para una precisión de un metro a escala urbana y reduce a la mitad el ancho de banda; (b) los *enumerados* de estado y tipo de vehículo se codifican en un solo byte en lugar de como cadenas de texto.

Cabe preguntarse por qué un formato *ad hoc* y no un binario estándar autodescriptivo como *MessagePack* o *CBOR*. La razón es que la instantánea tiene un esquema fijo, conocido de

antemano por ambos extremos: un formato autodescriptivo repetiría en cada mensaje los nombres de campo y las marcas de tipo (precisamente la redundancia que se quiere eliminar). El precio es el acoplamiento: servidor y cliente deben compartir la misma versión del esquema, riesgo que mitigan el byte de versión y el discriminador de la cabecera.

El mensaje se compone de una cabecera y, tras ella, un registro por vehículo. El Cuadro 5.3 especifica el formato a nivel de byte. Todo el formato es *little-endian*. El primer byte actúa como discriminador de tipo: `0x01` identifica un *tick* binario y `0x7B` (el carácter «{») un mensaje JSON, de modo que el cliente distingue ambos sin ambigüedad por el primer octeto. Los enumerados se codifican como un entero de un byte: el estado (`idle` $= 0$, `moving` $= 1$, ..., `finished` $= 6$) y el tipo de vehículo (`car` $= 0$, `moto` $= 1$, `truck` $= 2$), con la tabla de correspondencia compartida por servidor y cliente.

Campo	Tipo	B	Descripción
Cabecera (18 bytes)			
`magic`	`u8`	1	Identificador de formato (`0x01`)
`version`	`u8`	1	Versión del protocolo (`0x01`)
`tick`	`u32`	4	Número de *tick*
`sim_time`	`f32`	4	Tiempo de simulación (s)
`chunk_index`	`u16`	2	Índice del fragmento
`chunk_total`	`u16`	2	Número de fragmentos del *tick*
`n_vehicles`	`u32`	4	Número de vehículos del fragmento
Registro por vehículo ($\sim$ 33 bytes, variable)			
`id_len`	`u8`	1	Longitud del identificador (UTF-8)
`id`	`bytes`	`id_len`	Identificador del vehículo
`lon`, `lat`	`f32`$\times 2$	8	Posición (WGS84)
`h`	`f32`	4	Rumbo (grados)
`v`	`f32`	4	Velocidad (m/s)
`a`	`f32`	4	Aceleración (m/s^2)
`edge_idx`	`u32`	4	Índice de la arista actual
`progress`	`f32`	4	Progreso en la arista $[0, 1]$
`status`	`u8`	1	Estado (entero)
`lane`	`u8`	1	Carril
`vtype`	`u8`	1	Tipo de vehículo (entero)

Cuadro 5.3: Disposición a nivel de byte del *tick* binario (*little-endian*). La cabecera de tamaño fijo precede a un registro por vehículo, de tamaño variable solo por la longitud del identificador. El cambio de JSON a este formato reduce $\sim 4\times$ el ancho de banda y el coste de serialización (§ 6.6)

5.6.2 Instrumentación de rendimiento

El sistema incluye una capa de instrumentación pensada para observar el rendimiento sin perturbarlo apreciablemente. Distingase del *motor de analíticas* de la § 5.5.3: aquella mide el coste del propio motor (latencia de *tick*, serialización, etc.), mientras que este calcula las métricas *de dominio* del tráfico simulado (tiempos de viaje, congestión, impacto de incidentes).

Ambos comparten el mismo registro y los mismos canales de exposición, pero responden a preguntas distintas. El registro de instrumentación mantiene tres familias de métricas en memoria:

Histogramas: Almacenados en un *búfer circular* (*ring buffer*) de tamaño fijo, que al llenarse sobrescribe las muestras más antiguas —lo que evita que la memoria crezca sin límite en sesiones largas—. Para cada fase medida (la física IDM, la de MOBIL, la serialización, el envío por red, etc.) se calculan bajo demanda la mediana, los percentiles 95 y 99, el máximo y la media.

Contadores: Valores que solo crecen a lo largo de la sesión, como el tráfico total enviado por red o el número de colisiones detectadas.

Indicadores: Reflejan el último valor instantáneo y, a diferencia de los contadores, pueden tanto subir como bajar: el número de clientes conectados, los vehículos activos en el bucle de física, etc.

La exposición ocurre por dos canales:

- **Formato Prometheus**, en `GET /metrics` (texto), compatible con una recogida periódica (*scrape*) estándar de Prometheus cada 5–15 s.
- **JSON legible**, en `GET /api/simulation/metrics`, con un esquema versionado y algunas métricas *derivadas* ya calculadas (tráfico por segundo, frecuencia de *tick* efectiva, coste de física por vehículo). Cómodo para la inspección manual y para los *scripts* de comparación de escenarios.

5.6.3 Resiliencia operativa y detección de tareas lentas

Por encima de esa instrumentación, el sistema incluye dos mecanismos de detección de regresiones:

1. **Detección de tareas lentas**: *asyncio* se configura para emitir un aviso cuando una tarea bloquea el bucle de eventos más de 50 ms [Ram22]. Complementa a las métricas, pues delata operaciones de E/S síncronas o intensivas en CPU dentro del bucle sin necesidad de un *perfilador* externo; en la práctica, esa señal ha sido la primera en aparecer ante regresiones puntuales.
2. **Registro de rendimiento**: cada 100 *ticks*, el motor escribe en el *registro* una instantánea de las métricas en JSON, lo que permite analizarlas *a posteriori* aunque el proceso termine sin volcarlas a una base de datos.

El último mecanismo de esta sección atiende al rendimiento del propio cómputo de física, que es la parte más cara de cada *tick*. Mientras la flota es pequeña, calcular la física de todos los vehículos en un solo hilo es suficiente y el coste de coordinar varios hilos no compensaría. Por eso, por debajo de cierto umbral de población, el motor delega el cálculo en un único

hilo auxiliar. A partir de ese umbral, en cambio, sí compensa repartir el trabajo entre varios núcleos, y el motor pasa a un *grupo de hilos*.

El reparto no puede hacerse al azar, porque la física de un vehículo depende de sus vecinos inmediatos —su líder, los coches del carril contiguo—. La solución es agrupar los vehículos por proximidad geográfica: se superpone al mapa una cuadrícula uniforme de 8×8 celdas y se forma un grupo de trabajo (*bucket*) con los vehículos de cada celda, de modo que los vecinos caen casi siempre en el mismo grupo y cada hilo puede procesarlo de forma independiente. Dos refinamientos completan el esquema. Primero, para que ninguna celda muy poblada se convierta en un grupo desproporcionado que retrase a todos los demás, cualquier celda demasiado poblada se subdivide. Segundo, los grupos se despachan de mayor a menor tamaño (la heurística *Longest Processing Time first*), lo que minimiza el tiempo total de pared del *tick*, ya que evita que un grupo grande quede para el final cuando los hilos ya están ociosos. El único punto en que dos hilos pueden necesitar tocar datos compartidos es una colisión entre vehículos de grupos distintos; esa mutación, poco frecuente, se protege con un único *lock* global, de modo que el resto del cómputo transcurre sin contención.

Esta elección de hilos en lugar de procesos solo es posible gracias a la versión *free-threaded* de CPython 3.14 (3.14t, PEP 703) [Gro23], que elimina el GIL (Global Interpreter Lock): el *lock* global con el que el intérprete habitual serializa cualquier cómputo CPU-*bound* aunque haya varios núcleos disponibles [Bea10]. Con el GIL activo, la única vía de paralelismo real son procesos separados (el modelo previo del proyecto, basado en *ProcessPoolExecutor*), pero cruzar la frontera entre procesos exige empaquetar y desempaquetar cada vehículo, un coste de comunicación que suponía una fracción apreciable del presupuesto del *tick*. Los hilos *free-threaded* ejecutan en paralelo de verdad sobre las mismas estructuras en memoria, sin empaquetado intermedio, que es precisamente lo que hace viable el esquema de *buckets* descrito.

La contrapartida es que la física opera sobre *memoria compartida*, de modo que el sistema escala *verticalmente* (con los núcleos de una única máquina) y no *horizontalmente*: el techo de rendimiento es el de un solo equipo (§ 6.6).

5.7 El sistema en ejecución

El ensamblaje de las seis iteraciones anteriores da como resultado un sistema operativo de extremo a extremo. Esta sección lo muestra funcionando como un todo, el resultado final del trabajo de diseño e implementación.

La Figura 5.7 muestra el cliente *Godot* durante una sesión típica con la red vial de Talavera importada y aproximadamente $2\,000$ vehículos en circulación. La interfaz, con la escena 3D en vista oblicua, muestra el estado de la simulación, el tiempo transcurrido, el número de vehículos, los FPS y los controles organizados en paneles en la parte inferior.

Figura 5.7: El cliente *Godot* durante una sesión típica: HUD completo, red vial de Talavera renderizada en 3D y $\sim 2\,000$ vehículos circulando

Detrás de esa imagen estática se cierra, varias veces por segundo, el bucle completo que recorren las seis iteraciones. En cada *tick* el motor avanza la física de toda la flota repartiéndola entre varios núcleos (§ 5.6.3), serializa el estado resultante en el formato binario (§ 5.6.1) y lo difunde por WS a los clientes conectados. Cada cliente lo deserializa en un hilo aparte, interpola entre los dos últimos estados conocidos (§ 5.5.4) y dibuja la flota completa en una sola pasada con MULTIMESH (§ 5.5.4). El operador ve así un movimiento fluido a 60 FPS aunque el servidor solo publique un estado cada 333 ms.

Sobre ese bucle, la interfaz no se limita a observar: el operador puede intervenir en caliente sin tocar la línea de comandos. Crear un incidente, cerrar un carril o delimitar una ZBE altera el estado del servidor en el siguiente *tick*, y la flota reacciona a la vista —recalculando rutas o desviándose— en cuestión de segundos.

5.8 Incidencias y desviaciones respecto a la planificación

Descrito el sistema, cabe contrastar cómo se construyó con cómo se planificó construirlo. La *línea base* prevista en el Capítulo 4 (§ 4.3) tenía la regularidad de un plan; el desarrollo real, como es habitual, se apartó de ella en varios puntos —casi siempre al materializarse los riesgos identificados en el Cuadro 4.3— que el diagrama de Gantt de la Figura 4.1 refleja.

El orden real de las iteraciones no fue estrictamente secuencial. El empuje inicial de enero y febrero priorizó cerrar cuanto antes el lazo completo extremo a extremo (iteraciones 1, 2, 3 y buena parte de la 5) antes que el control fino del tráfico. Como consecuencia, las iteraciones 4 (control y eventos) y 6 (rendimiento) se ejecutaron en abril, una vez existía un sistema funcionando sobre el que medir y al que incorporar esos mecanismos con conocimiento de causa.

La desviación de mayor calado fue el coste de escalar el rendimiento (riesgo R1). La primera versión integrada funcionaba correctamente, pero no pasaba de unos cientos de vehículos: la serialización JSON del estado de cada *tick* llegaba a bloquear el bucle de eventos y consumía una fracción apreciable del presupuesto de cómputo, hasta convertirse en el cuello de botella del servidor (§ 5.6.1). Resolverlo exigió una iteración entera —la sexta— dedicada en exclusiva a escalar: paralelización del cómputo de física, reducción del grafo vial y sustitución del JSON por un formato binario. Durante esta fase, el límite máximo de vehículos se elevó en varios órdenes de magnitud respecto a las cifras de uso real para poder forzar el sistema en pruebas de carga. El resultado fue un techo de unos $8\,000$ vehículos, por encima del dimensionamiento inicial de $4\,000$ a $6\,000$ (§ 6.6). No haber anticipado el peso de esta iteración es la principal lección de planificación del proyecto (§ 5.9).

Varias incidencias obligaron a corregir el modelo de tráfico (riesgo R6). Los comportamientos emergentes del modelo microscópico requirieron ramas de corrección específicas: la lógica de asignación de carriles y la de las intersecciones semaforizadas se rehicieron tras detectar comportamientos incorrectos, y la aparición de colisiones entre vehículos —no contemplada al principio— se resolvió tratando cada choque detectado por la física como un incidente de tipo accidente (§ 5.4). En la detección de estas regresiones resultó decisivo el aviso automático de tareas que bloquean el bucle de eventos, que en la práctica fue la primera señal en aparecer ante cada regresión de rendimiento (§ 5.6.3).

Algunos parámetros de diseño se ajustaron tras medir. El carácter incremental permitió tomar decisiones con datos en lugar de a priori. El ejemplo más claro es la frecuencia de *tick*, rebajada de los 5 Hz iniciales a los 3 Hz definitivos.

El alcance se recortó de forma deliberada. Bajo la presión de los riesgos R4 y R5, dos funcionalidades inicialmente consideradas se dejaron fuera: el cálculo cuantitativo de emisiones de CO_2, por exigir datos de calibración no disponibles (§ 7.3), y la discriminación de la ZBE por etiqueta ambiental de la DGT, sustituida por una restricción por tipo de vehículo (§ 5.4.5). Ambas se documentan como limitaciones reconocidas y líneas de trabajo futuro en el Capítulo 7.

5.9 Retrospectiva: balance del proceso

Cerrar el capítulo con una retrospectiva permite hacer balance del proceso que produjo el sistema recién descrito. El balance se ciñe a la dimensión metodológica (Capítulo 4); la evaluación del sistema en sí se reserva para el Capítulo 6.

Qué funcionó. Mantener `develop` siempre ejecutable y disponer de un sistema funcional desde muy pronto fue la decisión de mayor impacto: convirtió cada duda de diseño en algo medible y permitió decidir con datos —la frecuencia de *tick*, el formato binario— en lugar de por intuición. La instrumentación ligera del backend, y en particular el aviso de tareas lentas, dio un control continuo del rendimiento más útil que cualquier campaña formal de medición.

Y anclar cada unidad de trabajo en un *issue* enlazado a su rama aportó trazabilidad sin coste añadido.

Qué no funcionó. El rendimiento se subestimó: la iteración que lo abordó llegó tarde y concentró al final del proyecto un riesgo que debía haberse acotado antes.

Qué se haría distinto. Reservar desde la planificación inicial una iteración de rendimiento y un presupuesto explícito de escalado, en vez de descubrir el techo a posteriori; y congelar antes el alcance, en lugar de mantener el modelo de emisiones como aspiración hasta bien entrado el proyecto.

A pesar de estas desviaciones, el proceso iterativo cumplió su cometido esencial: garantizar en todo momento un sistema ejecutable y permitir corregir el rumbo con evidencia. Las desviaciones no comprometieron los objetivos del trabajo, sino que acotaron conscientemente su alcance.

Capítulo 6

Pruebas y validación

ESTE capítulo valida empíricamente el sistema presentado en el Capítulo 5. Se estructura en tres bloques: la estrategia de pruebas automatizadas que valida la implementación (§ 6.1–§ 6.5); las mediciones de rendimiento del backend y del cliente (§ 6.6–§ 6.7)[1]; y dos casos de estudio aplicados a la realidad de Talavera (la ZBE y un corte de viales como el de la inundación del Tajo de febrero de 2026) que ilustran el valor del sistema como herramienta de análisis *what-if* (§ 6.8–§ 6.9).

6.1 Estrategia de pruebas automatizadas

La validación del backend sigue el modelo clásico de pirámide de pruebas: una base amplia de pruebas unitarias sin base de datos (sustituyen ASYNCSESSION por dobles de prueba), una capa intermedia de 23 pruebas de integración que ejercitan la persistencia, la ingesta OSM, la reconstrucción del grafo y los *endpoints* contra una base PostGIS real en contenedor, y una cúspide de pruebas de carga (§ 6.6). En conjunto suman 482 pruebas en 27 ficheros, que cubren todos los subsistemas: física IDM/MOBIL, colisiones, geometría, semáforos e intersecciones, cálculo de rutas y zonas, ingesta OSM, motor de simulación, persistencia y analíticas. La batería completa **pasa en su totalidad** y se integra en Continuous Integration (CI) mediante *marcadores* que permiten filtrar las que requieren base de datos.

Como ejemplo del valor de la batería de pruebas: al rebajarse la frecuencia de *tick* de 5 a 3 Hz (§ 6.6.1), las pruebas detectaron de inmediato las aserciones que daban por supuesto el valor antiguo, el comportamiento esperado de una batería de regresión.

6.2 Validación del modelo IDM/MOBIL

Más allá de la corrección numérica de las fórmulas, interesa comprobar que el modelo reproduce cualitativamente los fenómenos del tráfico real. Se han verificado tres: la **formación de cola tras un semáforo en rojo** y su descarga ordenada al pasar a verde [TK13]; las **ondas de frenado *stop-and-go***, que el IDM propaga hacia atrás en flujo congestionado [THH00]; y

[1] Todas las cifras de rendimiento de este capítulo se han medido sobre el banco de pruebas descrito en la § 4.4.2 (Metodología) y son reproducibles con los *scripts* del repositorio: `scripts/bench_inproc.py` (y su línea base monohilo `bench_inproc_serial.py`), `scripts/bench_wire.py`, `scripts/bench_cases.py` y `scripts/bench_cases_live.py`.

los **cambios de carril**, con los vehículos rápidos desplazándose al carril izquierdo y el desalojo correcto de un carril cerrado. El primer fenómeno se aprecia en el diagrama espacio-tiempo de la Figura 6.1.

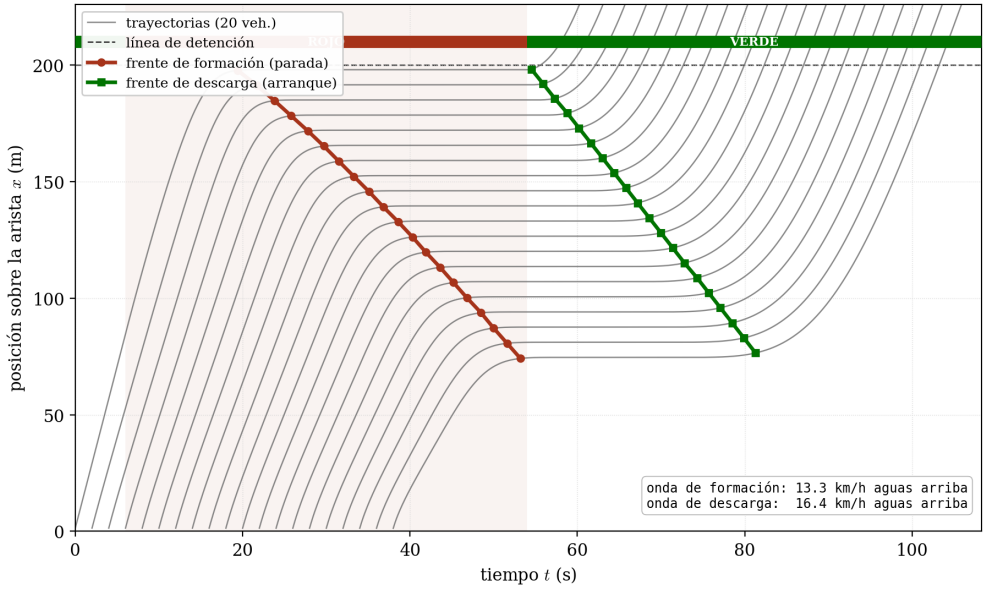

Figura 6.1: Diagrama espacio-tiempo de un pelotón de 20 vehículos atravesando un semáforo: el patrón clásico de cola con dos frentes de onda que se propagan aguas arriba, el de formación (parada, en rojo) y el de descarga (arranque, en verde). Eje X = tiempo, eje Y = posición sobre la arista

6.3 Validación del cálculo de rutas y las rotondas

El cálculo de rutas con A* y su integración con incidentes y zonas se valida en varios aspectos: que A* devuelve una alternativa cuando la ruta directa está bloqueada y, si todas lo están, devuelve igualmente la directa penalizada (el motor nunca se queda sin ruta, § 5.3.5); que una ZBE con reencaminamiento forzado desvía a los vehículos restringidos pero no a los demás, y que estos recuperan su ruta al desactivarla; y que en las rotondas se respeta la prioridad del anillo sin bloqueos. La Figura 6.2 muestra el efecto de un bloqueo sobre la ruta.

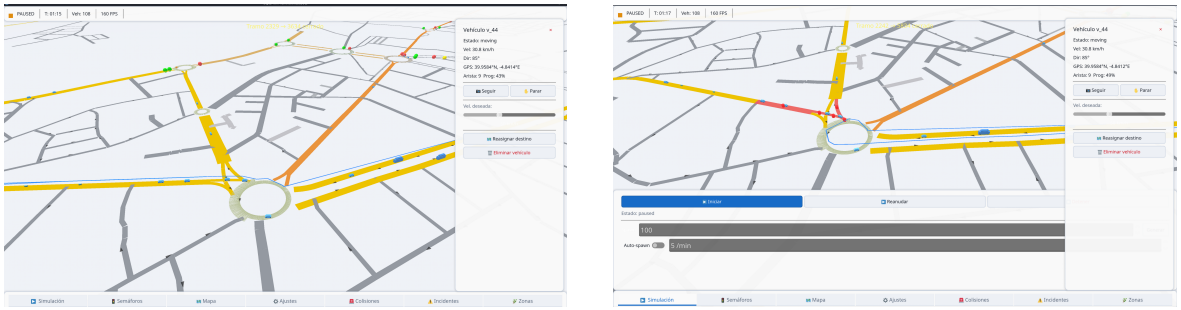

Figura 6.2: Misma pareja origen-destino sobre el mapa de Talavera: (izquierda) ruta calculada por A* sin ninguna arista bloqueada, que sigue el camino directo de menor coste; (derecha) ruta recalculada tras bloquear una arista de la anterior, que rodea la arista penalizada

6.4 Validación del control: semáforos, STOPs y rotondas

El control se valida en sus dos modos de operación: los ciclos automáticos de los semáforos (verde→ámbar→rojo, con el eje perpendicular complementario) y las anulaciones manuales (*overrides*), por nodo o globales. Una prueba comprueba una propiedad importante: forzar toda la red a rojo no convierte las intersecciones simples en barreras, porque a los nodos sin semáforo se les asigna verde por defecto (§ 5.4.2). Para los cruces no señalizados se valida la prioridad a la derecha y la detención efectiva ante un *stop*: el vehículo debe permanecer detenido un segundo antes de reanudar la marcha.

6.5 Validación del motor de analíticas

Las tres métricas del motor de analíticas (§ 5.5.3) se validan sobre grafos sintéticos, de forma aislada y determinista: el **tiempo de viaje** y la velocidad media frente a valores analíticos conocidos (descartando los viajes de duración nula); el **nivel de congestión** sobre una arista de capacidad conocida, comprobando que se contabiliza como saturada al superar el umbral; y el **impacto de incidentes**, verificando que solo se cuentan los vehículos cuya ruta restante atraviesa la arista bloqueada. Con esto se cierra la validación funcional; las secciones siguientes se ocupan del rendimiento.

6.6 Rendimiento del backend

Las secciones anteriores han validado que el sistema hace lo que debe; esta mide a qué *coste*. El rendimiento del backend se caracteriza por la estabilidad del *tick* bajo carga creciente (el escalado y el paralelismo *free-threaded*, § 6.6.2) y por el efecto del formato binario de transmisión (§ 6.6.3). Antes de los resultados se fijan el entorno y la metodología de medición.

6.6.1 Entorno y metodología de medición

La métrica clave del backend es la estabilidad del *tick* a $3\,\text{Hz}$ (un presupuesto de $333\,\text{ms}$ por *tick*) bajo carga creciente. La elección de $3\,\text{Hz}$ es el equilibrio entre exigencias contrapuestas: una frecuencia menor ($1\,\text{Hz}$) bastaría para el cómputo, pero produciría trayectorias visiblemente «a saltos» en el cliente; una mayor ($20\,\text{Hz}$) sería innecesaria para la dinámica vehicular y multiplicaría el coste por *tick* hasta comprometer el escalado. A $3\,\text{Hz}$, en cambio, el cliente suaviza el movimiento con un retardo de interpolación de $200\,\text{ms}$ (§ 6.7) sin que la latencia percibida resulte molesta, y el presupuesto de cómputo sigue siendo holgado. De hecho, la frecuencia se rebajó de un valor inicial de $5\,\text{Hz}$ a los $3\,\text{Hz}$ definitivos durante el desarrollo (§ 6.1).

Las mediciones se realizan con `scripts/bench_inproc.py`, que instancia el motor de simulación real (el mismo motor, generador de vehículos y difusor que sirve la API) y ejecuta el bucle de *tick* efectivo sin la indirección de la red HTTP. Para cada cardinalidad N se mantiene

la población constante reponiendo los vehículos que finalizan; se descarta un periodo de estabilización inicial de 20 s (la flota aún se puebla y las cachés se establecen) y se mide el régimen estacionario durante los 70 s siguientes, del orden de 210 *ticks* por punto. El Cuadro 6.1 resume los resultados.[2]

N **activos**	***tick*** **p50** (ms)	***tick*** **p95** (ms)	**física p50** (ms)	**ocupación** (×)	**serial. p50** (ms)	**B/veh**	**Hz efect.**
100	2,0	2,5	1,9	—	0,12	36,7	3,00
500	19,3	39,3	19,0	5,3	0,53	37,0	3,00
1 000	37,4	85,7	36,8	8,1	1,16	37,8	3,00
2 000	43,6	159,7	43,0	13,2	2,24	38,0	3,00
4 000	88,7	298,1	75,3	16,2	4,37	38,0	3,00
6 000	137,0	276,8	125,0	18,5	6,36	37,9	3,00
8 000	156,1	471,0	126,5	16,4	8,34	38,5	2,91
10 000	387,5	946,4	143,6	17,2	10,19	39,0	1,92

Cuadro 6.1: Tiempos de cómputo por ciclo medidos durante 70 s. Ciclo p50/p95 = mediana y percentil 95 del tiempo total del tick; «física» = tiempo de cómputo dedicado a los modelos IDM/MOBIL; «ocupación» = cociente entre el tiempo de CPU agregado de los hilos y el tiempo transcurrido real de la fase paralela (no aplicable por debajo del umbral de 500 vehículos); «serial.» = tiempo empleado en la serialización del estado del sistema (instantánea); «B/veh» = bytes por vehículo en el formato binario; «Hz efect.» = frecuencia de simulación sostenida. El límite temporal para mantener los 3 Hz es de 333 ms/ciclo.

6.6.2 Escalado y paralelismo *free-threaded*

La lectura del Cuadro 6.1 arroja cuatro conclusiones.

Primera: el coste computacional de la física presenta un escalado marcadamente sublineal. Al multiplicar la flota por ocho ($500 \to 4\,000$ vehículos), la mediana del tiempo de cálculo no experimenta el mismo crecimiento, sino que se multiplica por un factor inferior a cuatro ($19 \to 75$ ms). Asimismo, el incremento entre $1\,000$ y $2\,000$ vehículos es casi marginal ($37 \to 43$ ms). Este comportamiento se explica por el paralelismo: el procesamiento de cada tick se distribuye entre los 20 hilos lógicos compartiendo el espacio de memoria del proceso, gracias a la agrupación espacial de 8×8 descrita en la § 5.6.3.

Segunda: el aprovechamiento multinúcleo es efectivo y escala con la carga. La columna «ocupación» del Cuadro 6.1 es el cociente entre el tiempo de CPU agregado de los hilos trabajadores y el tiempo transcurrido real de la fase paralela. Esta métrica debe interpretarse como una medida de *utilización*, no como la aceleración real o *speedup*: una ocupación de $18{,}5\times$ con $6\,000$ vehículos indica que el cómputo llega a saturar los 20 hilos lógicos, pero no implica que el tick sea 18,5 veces más rápido. La aceleración real queda acotada por la línea

[2]Todas las cifras de esta sección proceden del banco de pruebas descrito en la § 4.4 de la Metodología; baste recordar los dos datos que condicionan su lectura: un *Intel Core i5-13600KF* de 20 hilos lógicos y *CPython* 3.14t *free-threaded*, con el GIL desactivado [Gro23].

base de ejecución monohilo (`scripts/bench_inproc_serial.py`, comparada de forma directa en la misma sesión). En concreto, el tiempo de la fase física se reduce de 107 a 57 ms con $2\,000$ vehículos ($1{,}9\times$) y de 340 a 76 ms con $4\,000$ ($4{,}5\times$). Aunque esta ganancia es netamente inferior a la tasa de ocupación, resulta decisiva. La mediana del tiempo de ciclo en un solo hilo para $4\,000$ vehículos (367 ms, equivalente a $2{,}4$ Hz efectivos) excede el límite temporal de 333 ms, mientras que la ejecución paralela logra mantener los $3{,}00$ Hz con un margen holgado. Además, esta comparativa constituye una prueba directa de que la versión *free-threaded* del intérprete neutraliza con éxito el GIL (§ 5.6.3). Bajo un entorno de ejecución estándar de Python, el reparto de carga en hilos no aportaría aceleración alguna en tareas limitadas por la capacidad de procesamiento (CPU-*bound*) [Bea10].

Tercera: el sistema mantiene una frecuencia de 3 Hz de forma holgada hasta los varios miles de vehículos. La mediana del tick se sitúa muy por debajo del límite de 333 ms incluso con cargas de $8\,000$ vehículos (156 ms), logrando sostener una frecuencia efectiva de $3{,}00$ Hz hasta los $\sim\ 6\,000$ vehículos. En el percentil 95 —que agrupa los ciclos de mayor carga computacional— el margen es más ajustado: en la franja de $4\,000$ a $6\,000$ vehículos, el *p95* se aproxima a los 300 ms. La aparente mejora de rendimiento entre los $4\,000$ y $6\,000$ vehículos que refleja el Cuadro 6.1 se debe a la variabilidad de muestreo intrínseca de los percentiles altos, no a un escalado real. De hecho, tres ejecuciones posteriores del banco de pruebas descartan esta anomalía (mostrando un *p95* que crece con la carga) y sitúan la fluctuación entre ejecuciones idénticas en torno al 15–$17\,\%$. En cualquier caso, la frecuencia efectiva no sufre una degradación notable hasta rebasar los $8\,000$ vehículos. Es al alcanzar los $10\,000$ vehículos cuando la propia mediana (388 ms) supera la ventana de ejecución y la frecuencia decae a $1{,}9$ Hz, estableciendo el límite operativo de la máquina de pruebas. Estos resultados respaldan y superan la capacidad estipulada en el diseño original ($4\,000$–$6\,000$ vehículos).

Cuarta: en situaciones extremas, el cálculo físico deja de ser el cuello de botella. Con $10\,000$ vehículos, la mediana del tiempo de cálculo de la física se sitúa en tan solo 144 ms (contenido gracias a la ejecución en paralelo), mientras que el tick completo se eleva a 388 ms. Esta discrepancia radica en las operaciones secuenciales del ciclo: la serialización del estado del sistema (cuyo coste crece linealmente con N, sumando hasta $\sim\ 10$ ms) y, de forma pronunciada, el procesamiento de los accidentes derivados de la densidad extrema de tráfico. Cada colisión exige registrarse como un incidente, lo que a su vez desencadena múltiples recálculos de ruta. En definitiva, ante densidades que superan ampliamente los casos de uso previstos, el factor limitante se traslada desde la simulación física de los vehículos hacia la gestión de eventos, un hallazgo clave para plantear futuras optimizaciones.

Estas cifras deben situarse frente a las plataformas de referencia del estado del arte (§ 3.2). La comparación directa es limitada: las cifras de rendimiento publicadas de SUMO miden la ejecución por lotes (completar la simulación tan rápido como permita el procesador), no la estabilidad de un bucle de tiempo real que difunde el estado a un cliente 3D en cada *tick*, y

su bucle de simulación es además monohilo [LBBW+18]. El resultado no debe leerse, por tanto, como «más rápido que SUMO», sino como la evidencia de que un motor a medida, paralelizado sobre memoria compartida, sostiene el régimen de *tiempo real* que este caso de uso exige, una métrica que dichas plataformas no reportan en los mismos términos.

Estos datos se representan gráficamente en la Figura 6.3.

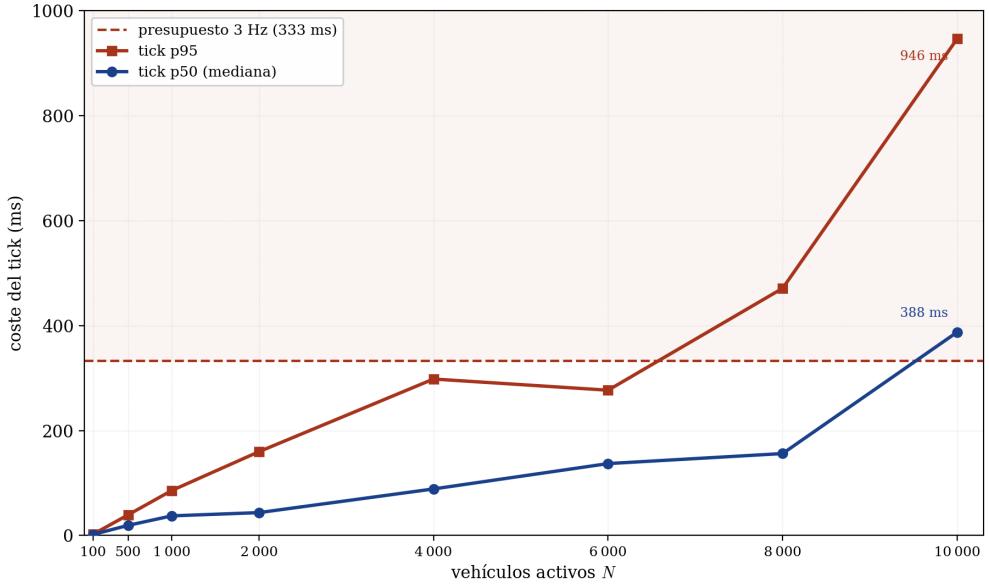

Figura 6.3: Coste del *tick* (mediana y percentil 95) frente al número de vehículos activos N, con la línea de presupuesto de 333 ms (3 Hz) en rojo.

6.6.3 Efecto del formato binario de transmisión

El otro factor determinante de la estabilidad del *tick* bajo carga es el formato binario de las instantáneas diseñado en la § 5.6.1, cuyo efecto se mide aquí de forma aislada con `scripts/bench_wire.py`, que serializa la misma flota en ambos formatos. La referencia JSON emplea el módulo `json` de la biblioteca estándar en su forma compacta (sin espacios), que es exactamente lo que usaba el protocolo original del prototipo. Para 2 000 vehículos, el formato binario produce una instantánea de 37,5 bytes por vehículo y tarda 1,5 ms en serializarse; la representación JSON equivalente ocupa 149,3 bytes por vehículo y tarda 5,9 ms. Es decir, el formato binario reduce el ancho de banda en un factor de 4,0× y el tiempo de serialización en 3,9×. Para descartar que la comparación se apoye solo en la opción más lenta, el banco incluye también *MessagePack* (`msgpack-python` con flotantes en precisión simple), un binario estándar *autodescriptivo*: aun eliminando la capa textual, sus 103,8 bytes por vehículo y 3,9 ms quedan 2,8× y 2,7× por encima del formato propio, porque repite en cada mensaje los nombres de campo y las marcas de tipo que el esquema fijo del *tick* hace innecesarios (§ 5.6.1). El Cuadro 6.2 recoge la comparación.

Es precisamente esta reducción la que mantiene la serialización (columna «serial.» del Cuadro 6.1) en el rango de los milisegundos (apenas 6 ms con 6 000 vehículos frente a los

Formato	Bytes/vehículo	Tamaño total	Serialización
JSON	149,3 B	298,6 KB	5,9 ms
MessagePack	103,8 B	207,7 KB	3,9 ms
Binario *ad hoc*	37,5 B	75,0 KB	1,5 ms
Mejora (vs. JSON)	4,0×	4,0×	3,9×

Cuadro 6.2: Serialización del estado de 2 000 vehículos en JSON (módulo `json` de la biblioteca estándar, forma compacta), *MessagePack* (`msgpack-python`, flotantes f32) y el formato binario de la § 5.6.1, medida con `scripts/bench_wire.py`. Frente a MessagePack, el formato propio reduce además 2,8× el tamaño y 2,7× el tiempo. El factor de $\sim 4\times$ frente a JSON es el que permite que la serialización ocupe una fracción mínima del presupuesto del *tick* (columna «serial.» del Cuadro 6.1) en lugar de competir con la física

~ 24 ms que costaría el JSON) y, por tanto, fuera del camino crítico mientras la física dispone del grueso del presupuesto.

6.7 Rendimiento del cliente

El rendimiento del cliente se rige por una propiedad verificable: gracias al renderizado por instancias (§ 5.5.4), el número de llamadas de dibujo no crece con el número de vehículos, lo que desacopla el coste de dibujado del tamaño de la flota. La latencia percibida entre el *tick* y su representación es, además, *constante y por diseño*: la fija el retardo de interpolación de 200 ms que el cliente introduce para suavizar el movimiento (§ 5.5.4), no el coste de cómputo, que es despreciable en comparación. El Cuadro 6.3 recoge una sesión en la que la flota se elevó por escalones hasta superar los 9 000 vehículos, anotando el régimen estacionario de cada escalón.

N (veh)	FPS medio	FPS p1	ll. dibujo	primitivos (M)	memoria (MB)	proc. cliente (ms)
$\sim$ 2 000	160	156	82	1,1	190	9,7
$\sim$ 4 000	155	136	82	1,9	199	10,0
$\sim$ 6 000	117	105	82	2,4	206	13,8
$\sim$ 8 000	86	76	71	3,2	217	20,0
$\sim$ 9 200	68	47	59	3,9	225	25,8

Cuadro 6.3: Rendimiento del cliente *Godot* frente al tamaño de la flota, medido con el PERFMONITOR (F3) sobre el banco de pruebas de la § 4.4.2 (GPU RTX 3070 Ti), con ventana maximizada a 3840×2160 y sincronía vertical a 160 Hz. FPS medio y percentil 1 (peor 1 % de fotogramas); «ll. dibujo» = llamadas de dibujo (órdenes enviadas a la GPU); primitivos = millones de primitivas renderizadas; memoria = memoria del proceso; «proc. cliente» = coste por *tick* de aplicar el estado recibido

Los datos confirman cuantitativamente la propiedad anunciada: las llamadas de dibujo se mantienen acotadas entre 59 y 82 mientras el número de vehículos se multiplica casi por cinco; el coste de *render* está, en efecto, desacoplado de N. La tasa de refresco se mantiene muy por encima de los 60 FPS hasta unos 8 000 vehículos (86 FPS de mediana, 76 en el

peor $1\,\%$), en notable simetría con el backend, que sostiene su *tick* a 3 Hz hasta esa misma cardinalidad (§ 6.6.2). Ambos extremos del sistema escalan de forma holgada hasta el orden de los $8\,000$ vehículos en el equipo de prueba. La memoria del proceso crece de forma modesta ($190 \rightarrow 225$ MB) y el coste de aplicar cada *tick* en el cliente permanece por debajo de la cadencia de fotograma a 60 FPS ($16{,}7$ ms) salvo en la franja más alta.

6.8 Caso de estudio: ZBE de Talavera de la Reina

El primer caso de estudio aplica el motor de simulación a un escenario real que la ciudad ha planteado pero aún no ha implantado: la ZBE de Talavera, cuya trayectoria normativa se recogió en la § 3.5.3. El perímetro de la ordenanza [Ayu25] define dos zonas —*Trinidad* y *Casco Histórico*— que abarcan en conjunto un ámbito de aproximadamente $0{,}625\,\text{km}^2$.

El experimento exige tres precisiones metodológicas previas. Primera: el tráfico empleado es *sintético* y no se ha calibrado con aforos reales (§ 7.3). Segunda: la geometría sí es la *real*, pues el polígono se ha trazado sobre el mapa reproduciendo el perímetro de la ordenanza. Como el grafo importado de OSM no conserva los nombres de las calles, la zona no se delimita enumerando viales, sino por su geometría. Tercera: el reparto de la restricción constituye deliberadamente una *cota superior* del efecto: restringir a la vez coches y camiones equivale a expulsar de la zona a la práctica totalidad del parque circulante (solo las motocicletas quedan exentas). Los resultados ilustran, en consecuencia, el método de evaluación de escenarios y la magnitud *relativa* del efecto en su caso más desfavorable, no la cifra absoluta sobre el tráfico real de Talavera.

El procedimiento experimental consta de cuatro pasos:

1. Se da de alta una ZBE con la geometría del perímetro real y régimen de aplicación `force_reroute` restringido a los tipos de vehículo `car` y `truck`: solo las motocicletas pueden cruzar. El gestor de zonas marca como restringidas las 304 aristas que el polígono contiene.
2. Se generan $3\,000$ pares origen-destino aleatorios entre nodos navegables de la red.
3. Para cada par se calcula la ruta *sin* restricción (la red libre, la que toma una moto) y la ruta de un coche o camión *con* la ZBE activa, usando el mismo A* del motor.
4. Se comparan ambas: cuántas rutas cruzan la zona, cuántas evita el tráfico restringido y cuánto rodeo adicional introduce la restricción.

El efecto de la zona sobre los flujos se aprecia en la comparativa de la Figura 6.4 y se cuantifica en el Cuadro 6.4.

La lectura de estas cifras: de las rutas que en condiciones normales atraviesan la zona ($38{,}9\,\%$ del total), la ZBE desvía fuera del polígono al $49{,}4\,\%$ de las afectadas, que asumen un rodeo medio de 700 m ($+32\,\%$); el cruce se reduce así a la mitad. El $19{,}7\,\%$ que aún atraviesa

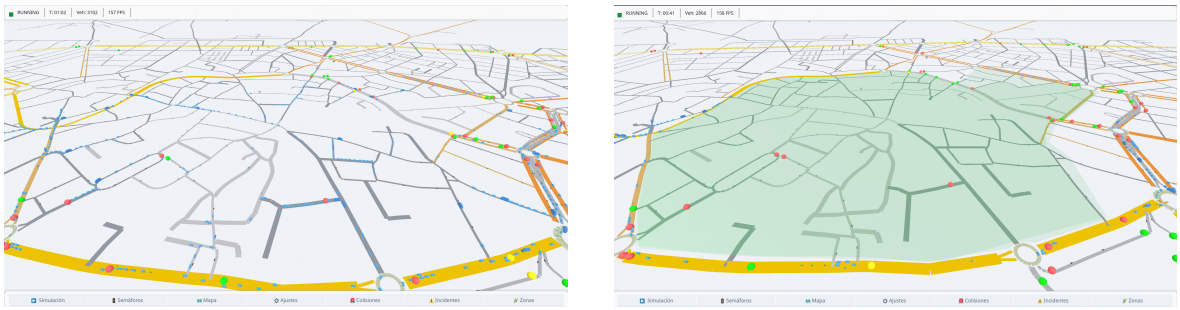

Figura 6.4: Mapa de Talavera con el perímetro real de la ZBE —zonas de *Trinidad* y *Casco Histórico*— superpuesto. (Izquierda) sin ZBE: los flujos de coches y camiones atraviesan la zona; (derecha) con ZBE activa: solo las motocicletas la cruzan, mientras coches y camiones la bordean por las arterias perimetrales

Métrica (sobre 3 000 rutas O-D)	**Valor**
Rutas que cruzan la zona sin restricción	38,9 %
Rutas de coche/camión que cruzan la zona con ZBE	19,7 %
Reencaminados de los que cruzaban	49,4 %
Rodeo medio del reencaminamiento	+700 m (+32,2 %)
Longitud media de ruta base	2 174 m

Cuadro 6.4: Efecto agregado de la ZBE restringida a coches y camiones (solo las motos cruzan) sobre 3 000 rutas origen-destino. La restricción `force_reroute` desvía a la mitad del tráfico que atravesaba la zona, a costa de un rodeo medio del 32 %; el resto carece de alternativa viable (origen o destino *dentro* de la zona) y la atraviesa con penalización

corresponde a desplazamientos con origen o destino dentro de la zona, sin alternativa que la evite, que el motor enruta igualmente penalizados (§ 5.3.5). Es exactamente el comportamiento esperado de una ZBE *permisiva* con acceso de residentes (y de motocicletas, exentas), y cuantifica el método: el simulador no solo dice *si* la zona desvía tráfico, sino *cuánto* y a costa de qué rodeo.

6.9 Caso de estudio: corte viales por inundación

El segundo caso de estudio reproduce el episodio que la crecida del Tajo provocó a su paso por Talavera a comienzos de febrero de 2026: la subida del río —en nivel naranja— anegó garajes y obligó a cortar al tráfico varias calles, además de activar el Centro de Coordinación Operativa Municipal (CECOPAL) [Caz26, Cas26]. Este episodio constituye un escenario *what-if* especialmente adecuado para el simulador precisamente porque cierra viales por las que circulan vehículos, alterando de forma directa el cálculo de rutas sobre la red.

Con la misma salvedad metodológica de la sección anterior (tráfico sintético, viales identificados por geometría y no por nombre), el procedimiento corta las 13 calles *realmente* anegadas durante el episodio. Cada calle se indica por su tramo (par de puntos entrada-salida en coordenadas) y se empareja con la red (*map-matching*) seleccionando, mediante una consulta espacial PostGIS, las aristas cuya geometría discurre a menos de 20 m del tramo; el corte

abarca así 104 aristas. Sobre esa red dañada se mide el efecto en las 3 000 rutas origen-destino. La Figura 6.5 muestra el HUD durante el escenario y el Cuadro 6.5 recoge las cifras.

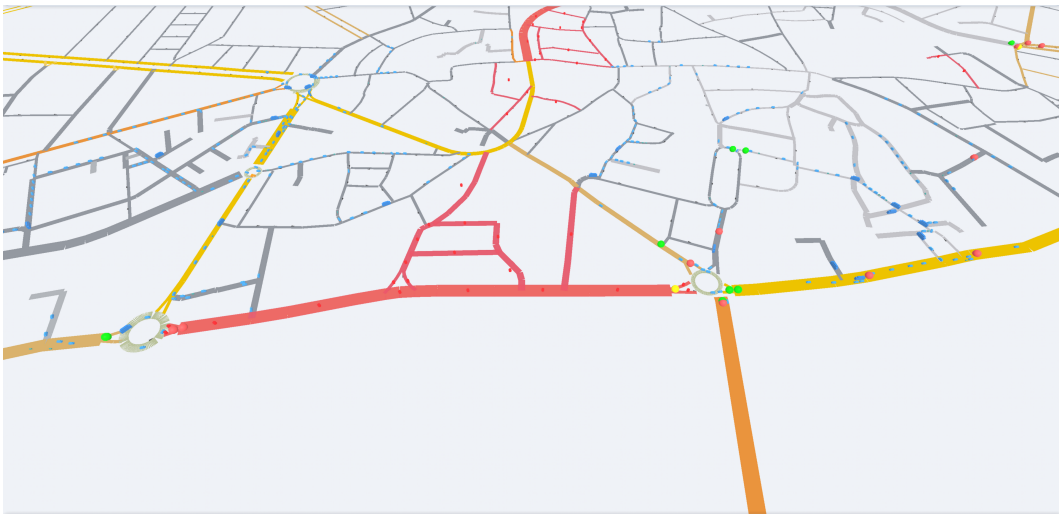

Figura 6.5: HUD durante el episodio: el mapa resalta en rojo las aristas afectadas y se observan las rutas de varios vehículos recalculándose en vivo y saturando las arterias perimetrales

Métrica (sobre 3 000 **rutas O-D,** 13 **calles /** 104 **aristas)**	**Valor**
Rutas que usaban alguna calle cortada	60,1 %
Reencaminadas con éxito	100 %
Rutas sin alternativa posible	0
Que aún cruzan (origen/destino en calle cortada)	225
Rodeo medio del reencaminamiento	+681 m (+31,3 %)

Cuadro 6.5: Efecto agregado de cortar las 13 calles realmente anegadas. Seis de cada diez rutas se ven afectadas, pero *todas* encuentran un camino alternativo. El rodeo medio es notable (+681 m, +31 %) porque varias de las calles cortadas están en el casco y concentran tráfico de paso; aun así, las 225 rutas que aún atraviesan un corte tienen su origen o destino en la propia calle anegada. Los valores miden el primer transitorio de redistribución, no el equilibrio a largo plazo

El contraste con el caso de la ZBE es instructivo. El corte por inundación afecta a más rutas que la ZBE (60,1 % frente a 38,9 %) pese a inhabilitar menos aristas (104 frente a 304): las calles anegadas se reparten por la red e incluyen viario de paso, mientras la ZBE es una mancha contigua y central. El rodeo medio, en cambio, resulta de magnitud *similar* en ambos casos (+31 % en la inundación, +32 % en la ZBE). La diferencia cualitativa está en la *conectividad*: ninguna ruta queda sin alternativa en ninguno de los dos escenarios (§ 6.3), de modo que el coste de ambas perturbaciones se manifiesta como rodeo adicional, no como desconexión.

Capítulo 7

Conclusiones y trabajo futuro

ESTE capítulo final cierra el TFG: valora el cumplimiento de los objetivos, las contribuciones del trabajo, sus limitaciones reconocidas y las líneas de trabajo futuro. Recoge después las competencias de la intensificación que el trabajo acredita y termina con una reflexión final.

7.1 Cumplimiento de los objetivos

El objetivo general planteado en el Capítulo 2 era:

> Desarrollar un gemelo digital modular para la simulación microscópica, visualización interactiva 3D y control en tiempo real del tráfico urbano de Talavera de la Reina, que permita evaluar escenarios *what-if* y comparar políticas de gestión de tráfico.

El sistema entregado responde al objetivo general: reconstruye automáticamente la red real de Talavera, sostiene su tráfico microscópico en tiempo real y permite al operador materializar en segundos los escenarios *what-if* (activación de la ZBE, corte de calles por inundación, incidente en arteria) sobre una visualización 3D interactiva.

Los seis objetivos parciales se consideran cumplidos.La trazabilidad completa entre cada objetivo, su sección de implementación y su sección de validación se recoge en el Cuadro 7.1.

Obj.	Enunciado (resumido)	Sección	Estado
1	Arquitectura del sistema y comunicación	§ 5.1	Completo
2	BD geoespacial e ingesta OSM	§ 5.2, § 5.7	Completo
3	Modelo vehicular, gestor de vehículos, cálculo de rutas	§ 5.3, § 6.2, § 6.3	Completo
4	Control de eventos: semáforos, incidentes, carriles, zonas	§ 5.4, § 6.4, § 6.8	Completo
5	API REST + transmisión WS + analíticas	§ 5.5, § 6.5, § 6.6	Completo
6	Renderizado, cámaras, panel de control, visualización de eventos	§ 5.5.4, § 6.7	Completo

Cuadro 7.1: Mapeo de los seis objetivos parciales del enunciado a sus secciones de implementación y resultados. Las limitaciones reconocidas se discuten en § 7.3

7.2 Contribuciones

El TFG aporta, más allá del cumplimiento literal de los objetivos:

1. **Un simulador de código abierto completo**, publicado en repositorios públicos y reproducible por un tercero siguiendo el Anexo B.
2. **Integración fina entre OSM y microsimulación** mediante un ETL que detecta los puntos de corte de la red y construye el grafo apto para IDM/MOBIL. El sistema es **independiente del municipio**: basta sustituir el fichero OSM de entrada para simular cualquier ciudad, como se ha comprobado con Toledo.
3. **Parametrización de IDM/MOBIL** con tres perfiles de vehículo (turismo, motocicleta y camión) razonable para entornos urbanos españoles y modificable en caliente.
4. **Modelado de ZBE configurable**, con tres modos de control que cubren tanto el régimen estricto como el «no sancionador» de la ordenanza inicial de Talavera.
5. **Formato binario de transmisión** para el estado de cada *tick*, que reduce unas cuatro veces el ancho de banda y el coste de serialización frente a JSON.
6. **Cliente 3D con renderizado por instancias** sobre *Godot*, capaz de dibujar miles de vehículos a 60 FPS con un movimiento interpolado coherente con el servidor.
7. **Dos casos de estudio aplicados a Talavera**: la ZBE y el corte de calles por la inundación del Tajo, ambos derivados de eventos reales recientes de la ciudad.

7.3 Limitaciones reconocidas

El presente trabajo presenta seis limitaciones que se reconocen aquí de forma explícita.

Cálculo cuantitativo de emisiones de CO_2. El sistema modela el efecto operacional de una ZBE (denegación de generación, recálculo forzado de rutas y restricciones por tipo de vehículo,

§ 7.3) pero no integra un modelo cuantitativo de emisiones que permita reportar, por ejemplo, que la ZBE estricta reduce las emisiones de CO_2 en un porcentaje determinado dentro del perímetro. Esa magnitud requiere un modelo del tipo COPERT [NGKS09], HBEFA [INF19] o MOVES [U.S20] alimentado con la velocidad instantánea, la aceleración y el tipo de cada vehículo simulado.

La omisión es deliberada: calibrar un modelo de emisiones para Talavera exige aforos por calle y un censo del parque vehicular, ninguno de los cuales es público —sin esos datos, cualquier valor absoluto de toneladas de CO_2 carecería de soporte empírico—. La propuesta de implementación se desarrolla en la § 7.4.

Restricción de la ZBE por tipo de vehículo, no por etiqueta. La restricción de la ZBE implementada discrimina por *tipo* de vehículo (`coche`, `camión`, `motocicleta`), que es la unidad de restricción que maneja el prototipo (§ 5.4.5). Constituye así una aproximación a la discriminación real por etiqueta ambiental de la DGT (§ 3.5.2). Modelar la etiqueta concreta de cada vehículo exigiría añadir ese atributo al modelo de vehículo y al protocolo binario. Queda ligada al motor de emisiones de la § 7.3 y se recoge como trabajo futuro en la § 7.4.

Calibración con aforos reales. Los casos de estudio del Capítulo 6 emplean tráfico sintético, no calibrado con aforos reales. Talavera no publica las intensidades medias diarias por calle, pese a haberlas medido el PMUS de Doymo en 2023 [Doy23]. Como consecuencia, el sistema evidencia el método de evaluación de escenarios sin pronunciarse sobre la magnitud absoluta de los efectos. La propuesta de calibración con aforos reales se desarrolla en la § 7.4.

Cliente Godot sin pruebas automatizadas. El backend tiene una cobertura de pruebas amplia, pero el cliente *Godot* no incorpora ninguna batería automatizada; su validación se ha hecho por inspección visual. La batería Godot Unit Test (GUT) sobre esa lógica de dominio se concreta como trabajo futuro (§ 7.4).

Métricas no persistidas a base de datos. La instrumentación mantiene las métricas en memoria y las expone bajo demanda (en formato Prometheus y JSON), pero no persiste series temporales en una base de datos. Para el análisis fuera de línea basta con la recogida externa de esas métricas o con el *registro* estructurado que el motor emite periódicamente. La persistencia nativa se reserva como trabajo futuro.

Resultados de rendimiento ligados al hardware de prueba. Las cifras de rendimiento del Capítulo 6 (§ 6.6) son específicas del equipo de prueba empleado (14 núcleos físicos y 20 hilos lógicos). El techo de $\sim 8\,000$ vehículos escala con el número de núcleos disponibles, pero no debe extrapolarse sin más a otro *hardware*. Las mediciones cuantifican el comportamiento sobre una máquina concreta y la *tendencia* de escalado, no una garantía absoluta de capacidad.

7.4 Trabajo futuro

Las limitaciones anteriores apuntan a una *hoja de ruta* en cuatro bloques.

Modelo de emisiones COPERT/HBEFA. La integración de un modelo cuantitativo de emisiones (tipo COPERT [NGKS09]) es la línea de mayor impacto. Como cada vehículo conoce su tipo y su velocidad instantánea, el motor podría integrar la emisión a lo largo de la simulación y agregarla por arista y por zona. El verdadero coste estaría en la calibración (la distribución del parque vehicular por categoría y norma Euro). Ese mismo atributo ambiental por vehículo permitiría además aplicar la ZBE *por etiqueta* de la DGT, y no solo por tipo de vehículo como en el prototipo (§ 5.4.5).

Calibración con aforos del PMUS. La segunda línea consiste en solicitar formalmente los aforos por calle del PMUS de Doymo de 2023 [Doy23] y emplearlos para calibrar el patrón de generación de vehículos.

Demanda basada en actividades. La tercera línea consiste en enriquecer el modelo de demanda. La generación actual produce vehículos con orígenes y destinos uniformemente distribuidos, lo cual resulta razonable para el régimen estacionario pero no captura los picos de hora punta ni los patrones laborales. La aproximación natural es seguir el modelo de MATSim [HNA16]: cada agente sintético dispone de una agenda diaria de actividades (trabajo, estudio, ocio) y los desplazamientos se modelan como transiciones entre actividades.

Pulido operativo. La quinta línea es de carácter operativo más que conceptual:

- Almacenamiento persistente de las métricas en TimescaleDB mediante políticas de retención configurables, en sustitución del actual archivo de registro JSON.
- Bastionado (*hardening*) de la seguridad del servicio. Esto abarca la implementación de autenticación basada en *tokens*, autorización fundamentada en roles, control de la tasa de peticiones (*rate limiting*) y una política Cross-Origin Resource Sharing (CORS) restrictiva. Estas medidas se encuentran actualmente fuera del alcance del proyecto, dado que el sistema se ha concebido para su ejecución local o bajo una red de confianza (§ 5.5), pero constituyen un requisito ineludible previo a cualquier despliegue público.
- Desarrollo de una batería de pruebas unitarias mediante GUT para el cliente desarrollado en Godot. Estas pruebas se centrarán en los gestores de estado y los conversores de coordenadas; es decir, en aquella lógica de negocio que resulta verificable de forma independiente al motor de renderizado.

7.5 Competencias de la intensificación en Sistemas de Información

El TFG se enmarca en la intensificación de *Sistemas de Información* del Grado en Ingeniería Informática. El Cuadro 7.2 relaciona las competencias específicas de esa tecnología específica

que el trabajo acredita con la evidencia concreta —secciones de esta memoria— que las respalda.

Comp.	Enunciado (resumido)	Evidencia en el TFG
SI3	Participar en la especificación, diseño, implementación y mantenimiento de los sistemas de información y comunicación	Cubre el ciclo completo del sistema: requisitos (Cap. 2), diseño de la arquitectura cliente-servidor y su canal de comunicación en tiempo real (§ 5.1), implementación de los seis subsistemas (Cap. 5) y su validación (Cap. 6)
SI1	Integrar soluciones de tecnologías de la información y las comunicaciones y procesos para satisfacer las necesidades de información de las organizaciones de forma efectiva y eficiente	Integración de fuentes heterogéneas mediante el ETL desde OSM (§ 5.2) y explotación de la información para la toma de decisiones de movilidad municipal (§ 6.8, § 6.9)
SI2	Determinar los requisitos de los sistemas de información y comunicación atendiendo a la seguridad y al cumplimiento de la normativa vigente	Determinación de requisitos de datos, tiempo real, rendimiento y persistencia (Cap. 2, § 5.1); uso de datos cartográficos abiertos con su licencia y encaje en la normativa de la ZBE (§ 5.4.5)

Cuadro 7.2: Competencias específicas de la intensificación de Sistemas de Información que el TFG acredita, con la evidencia que las respalda en la memoria

7.6 Reflexión final

La construcción de un gemelo digital de tráfico para una ciudad mediana integra ingeniería de sistemas, matemática aplicada, ingeniería gráfica y conocimiento de dominio en urbanismo y normativa de movilidad. Ninguno de los cuatro elementos es novedoso por sí mismo, y el valor del trabajo reside en su combinación.

Aprendizajes técnicos. Destacan tres: el cuidado que exige el bucle de eventos asíncrono bajo carga (su detección de bloqueos delató más regresiones de rendimiento que cualquier *perfilador*); el coste real de la serialización JSON en el camino crítico, que convirtió el formato binario en un requisito de ingeniería y no en una optimización opcional; y que el renderizado masivo en 3D no es automático, sino fruto de una gestión cuidadosa de la geometría.

Aprendizajes metodológicos. La inversión más rentable fue el *registro* estructurado que el motor emite periódicamente, que mantuvo el rendimiento bajo control; y anclar los casos de estudio en eventos reales y recientes de la ciudad convirtió una demostración técnica en una herramienta con relevancia inmediata.

Aprendizajes profesionales. El gobierno abierto municipal todavía no traslada al ciudadano las cifras de movilidad: se invierte en infraestructura de ZBE sin publicar apenas datos de tráfico urbano. Ese vacío puede constituir un nicho legítimo para herramientas como la de este TFG, que permiten a un técnico —o a un ciudadano— plantear en minutos preguntas concretas sobre el efecto de cada decisión.

ANEXOS

Anexo A

Declaración del uso de herramientas de IA generativa

Durante la realización de este TFG se han empleado herramientas de inteligencia artificial generativa como apoyo al trabajo del autor. En aras de la integridad académica, se declara aquí de forma responsable el alcance de ese uso.

Estas herramientas se han utilizado con tres finalidades:

- **Redacción y revisión de la memoria.** Mejora de la redacción y el estilo, corrección ortográfica y gramatical, reorganización de contenidos y síntesis de borradores propios. Las ideas, el contenido técnico, los datos, los resultados y las conclusiones son del autor.
- **Asistencia en la programación.** Generación y refactorización de fragmentos de código del *backend* en Python y del cliente en Godot, y explicación de conceptos y de mensajes de error. Todo el código se ha revisado, probado e integrado por el autor.
- **Depuración y pruebas.** Ayuda en el diagnóstico de errores, en la interpretación de las trazas de ejecución y en la elaboración de pruebas automatizadas.

El autor ha revisado, verificado y validado todo el material elaborado con apoyo de estas herramientas, y asume la plena responsabilidad del contenido final de esta memoria y del software desarrollado. Las herramientas de IA generativa no se han empleado para inventar datos, resultados ni referencias bibliográficas.

LOPEZ MARIN ISMAEL - 50636404H

Firmado por: LOPEZ MARIN ISMAEL - 50636404H

Fecha: 2026-06-12 17:43:32 CEST

Anexo B

Manual de configuración y despliegue

Este anexo recoge el procedimiento completo para configurar, desplegar y poner en marcha el sistema, partiendo de una máquina limpia. Su objetivo es la **reproducibilidad**: que un tercero pueda levantar el gemelo digital —el *backend* de simulación, la base de datos y el cliente de visualización— y reproducir los resultados del Capítulo 6 sin más información que la aquí contenida. El despliegue de referencia es el de la máquina de desarrollo (§ 4.4.2), bajo Linux; el uso de contenedores hace, no obstante, que el procedimiento sea portable a cualquier sistema operativo con Docker.

El sistema se reparte en tres repositorios bajo la organización `DigitalTwinTalavera`: el *backend* en Python (`tfg-dt-python-backend`), el cliente 3D en Godot (`tfg-dt-godot-client`) y la propia memoria en LaTeX (`tfg-dt-memory`). Para desplegar el sistema solo intervienen los dos primeros. A lo largo del anexo se ofrecen dos vías para el *backend*: una basada en **Docker** —recomendada por su reproducibilidad— y otra de **instalación manual**, útil para desarrollo o para entornos sin contenedores.

B.1 Requisitos previos

Hardware. El sistema se desarrolló y midió sobre la máquina descrita en la § 4.4.2 (Intel Core i5-13600KF, 31 GiB de RAM, Linux). No se requiere ese equipo para ejecutarlo: el *backend* funciona en cualquier máquina x86-64 con Docker, si bien el cómputo de física es CPU-*bound* y escala con el número de núcleos, por lo que un procesador multinúcleo es lo que permite sostener varios miles de vehículos. El cliente exige una GPU compatible con el renderizador *Forward+* de Godot para alcanzar los 60 FPS de la visualización. Como orientación, 8 GiB de RAM bastan para escenarios modestos y 16 GiB resultan holgados para los casos de estudio del Capítulo 6.

Software. El Cuadro B.1 enumera las herramientas necesarias, indicando si lo son para la vía con Docker, para la instalación manual o para ambas. Todas son de código abierto.

Herramienta	Versión	Vía	Uso
Git	2.x	Ambas	Clonado de los repositorios
Docker Engine	24 o superior	Docker	Contenedorización del *backend* y la base de datos
Docker Compose	v2 (*plugin*)	Docker	Orquestación de los servicios (`docker compose`)
Godot Engine	4.6	Ambas	Cliente de visualización 3D (renderizador *Forward+*)
Python	3.14 (`3.14t`)	Manual	Intérprete del *backend*; se recomienda la variante *free-threaded* (PEP 703)
PostgreSQL	15 o superior	Manual	Base de datos relacional
PostGIS	3.3	Manual	Extensión geoespacial de PostgreSQL
uv	reciente	Manual	Instalación del intérprete `3.14t` y de las dependencias (opcional pero recomendado)

Cuadro B.1: Herramientas necesarias para el despliegue, según la vía elegida para el *backend*

B.2 Arquitectura de despliegue

El sistema sigue una topología cliente-servidor de tres piezas: el cliente Godot, el *backend* y la base de datos. En la vía con Docker, el *backend* y la base de datos —junto con la herramienta de administración *pgAdmin*— se ejecutan como contenedores en un mismo *host*, mientras que el cliente Godot corre de forma nativa en la máquina del usuario. La Figura B.1 resume esta disposición y los canales de comunicación entre las piezas, y el Cuadro B.2 recoge los puertos que el sistema expone.

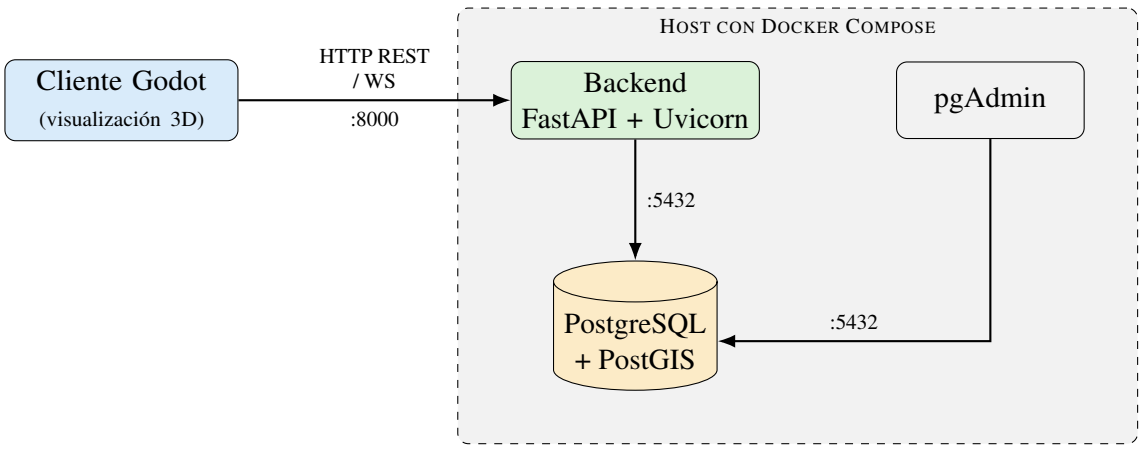

Figura B.1: Topología de despliegue. El cliente Godot accede por HTTP/WS (puerto 8000) al *backend*, que a su vez consulta la base de datos PostGIS (puerto 5432). *pgAdmin* es opcional y solo sirve para administrar la base de datos

Puerto	Servicio	Descripción
8000	*Backend* (FastAPI)	API REST de comandos y canal WS de estado en tiempo real
5432	PostgreSQL + PostGIS	Base de datos espacial; la consulta el *backend*
5050	*pgAdmin*	Interfaz web de administración de la base de datos (opcional)

Cuadro B.2: Puertos que expone el despliegue por defecto y servicio asociado a cada uno

B.3 Despliegue del *backend* con Docker

Es la vía recomendada: levanta la base de datos y el *backend* con una sola orden y de forma reproducible, sin necesidad de instalar Python ni PostgreSQL en el *host*. El primer paso es clonar el repositorio y crear el fichero de configuración a partir de la plantilla:

```
git clone https://github.com/DigitalTwinTalavera/tfg-dt-python-backend.git
cd tfg-dt-python-backend
cp .env.example .env        # variables de entorno (ver Cuadro de referencia)
```

Los valores por defecto de `.env` sirven para un despliegue local; en cualquier entorno no aislado deben cambiarse al menos las credenciales de la base de datos (`POSTGRES_PASSWORD`). A continuación se construyen las imágenes y se levantan los servicios en segundo plano:

```
docker compose up -d --build   # docker-compose up -d --build  (Compose v1)
```

La orquestación (`docker-compose.yml`) define tres servicios —`postgres`, `backend` y `pgadmin`— en una red interna común. El servicio `backend` declara una dependencia con condición de salud sobre `postgres`, de modo que no arranca hasta que la base de datos responde a `pg_isready`; así se evita la condición de carrera típica del primer arranque. La imagen del *backend* (`Dockerfile`) parte de *uv* e instala un *CPython* 3.14 *free-threaded* (`3.14t`, PEP 703), necesario para el paralelismo de la física (§ 4.4.1). En su arranque, el contenedor ejecuta automáticamente las migraciones de esquema antes de iniciar el servidor:

```
alembic upgrade head && uvicorn app.main:app --host 0.0.0.0 --port 8000 \
        --ws-max-size 4194304
```

En la primera ejecución, el contenedor de PostgreSQL aplica además el guion `scripts/init.sql`, que habilita las extensiones `postgis` y `postgis_topology`. Para comprobar que todo está en marcha basta consultar el *health check* detallado, que informa también del estado de la base de datos:

```
curl http://localhost:8000/api/health/detailed
```

Una respuesta con estado correcto confirma que el *backend* y la base de datos están operativos. La administración de la base de datos queda disponible en `http://localhost:5050` (*pgAdmin*), con las credenciales fijadas en `.env`. Para detener los servicios se emplea `docker compose down`; añadiendo `-v` se eliminan también los volúmenes de datos.

B.4 Carga de la red vial

Tras desplegar el *backend*, la base de datos está vacía: hay que importar la red vial de Talavera de la Reina desde OSM. El proceso tiene dos pasos: obtener el extracto cartográfico y cargarlo mediante el proceso ETL.

Obtención del extracto. Para un área reducida, lo más cómodo es *Overpass Turbo*: centrando el mapa en Talavera y exportando el resultado de la consulta del Listado siguiente como datos OSM en bruto al fichero `data/talavera.osm`.

```
[out:xml][timeout:300];
(
  way["highway"~"^(motorway|trunk|primary|secondary|tertiary|residential|service)"]
    ({{bbox}});
  node(w);
);
out body;
>;
out skel qt;
```

De forma alternativa, para regiones mayores puede descargarse el extracto de Castilla-La Mancha de *Geofabrik* y recortar el área de Talavera con *osmium*, indicando la caja envolvente de la ciudad:

```
osmium extract -b -4.90,39.88,-4.78,39.98 \
  castilla-la-mancha-latest.osm.pbf -o data/talavera.osm
```

Importación. El extracto se carga con el guion de línea de órdenes incluido en el *backend*, que ejecuta el proceso ETL (extracción de nodos y aristas, filtrado de tipos de vía y construcción del grafo) y los persiste en la base de datos:

```
python scripts/load_osm.py data/talavera.osm --clear
```

La opción `--clear` vacía los datos previos antes de importar. La misma operación puede lanzarse a través de la API REST, útil cuando el *backend* corre en contenedor:

```
curl -X POST http://localhost:8000/api/map/import \
  -H "Content-Type: application/json" \
  -d '{"filepath": "talavera.osm", "clear_existing": true}'
```

Por último, si se desea que el *backend* cargue el mapa de forma automática en su arranque cuando la base de datos esté vacía, basta indicar el fichero en la variable `MAP_FILE` de `.env` (ruta relativa a `data/`).

B.5 Configuración y ejecución del cliente

El cliente es un proyecto de Godot 4.6 que se ejecuta de forma nativa. Tras clonar su repositorio, se abre desde el gestor de proyectos de Godot importando la carpeta del proyecto; el motor importará los recursos en la primera apertura.

```
git clone https://github.com/DigitalTwinTalavera/tfg-dt-godot-client.git
```

Toda la configuración del cliente está centralizada en `config/config.gd`. Si el *backend* no corre en `localhost:8000`, deben ajustarse las constantes de conexión, a partir de las cuales se derivan automáticamente las Uniform Resource Locator (URL) de la API y del canal WS:

```
const BACKEND_HOST: String = "localhost"
const BACKEND_PORT: int = 8000
const BACKEND_PROTOCOL: String = "http"
```

Con el *backend* en marcha y la red vial importada, se ejecuta el proyecto con `F5` (o el botón de reproducción). Se lanza la escena principal (`scenes/main.tscn`), desde la que se verifica la conexión y se carga la red para su visualización 3D.

B.6 Verificación del despliegue

Una puesta en marcha correcta puede comprobarse siguiendo, en orden, estos pasos:

1. ***Backend* y base de datos.** La orden `curl http://localhost:8000/api/health/detailed` devuelve un estado correcto e informa de la conexión con la base de datos.
2. **Red vial cargada.** El extremo `/api/map/import/status` (o una consulta a `/api/map/nodes`) confirma que existen nodos y aristas en la base de datos.
3. **Cliente.** Al ejecutar el cliente, la verificación de conexión (*health check*) debe resultar

exitosa y la carga de la red debe poblar la escena 3D con la red de Talavera.

4. **Simulación.** Iniciada la simulación, el cliente recibe por WS el flujo de estado y los vehículos se mueven sobre la red, lo que confirma que las tres piezas se comunican correctamente.

B.7 Referencia de parámetros de configuración

La configuración del *backend* se gobierna por variables de entorno (fichero `.env`), validadas al arranque por *Pydantic*. El Cuadro B.3 resume las más relevantes. La del cliente reside en `config/config.gd`; el Cuadro B.4 recoge las constantes que con más frecuencia requieren ajuste.

Variable	Por defecto	Descripción
`PORT`	`8000`	Puerto en que escucha el *backend*
`HOST`	`0.0.0.0`	Interfaz de escucha del servidor
`DEBUG`	`true`	Modo de depuración
`LOG_LEVEL`	`INFO`	Nivel de registro (`DEBUG` … `CRITICAL`)
`CORS_ORIGINS`	`["*"]`	Orígenes permitidos para CORS
`MAP_FILE`	(vacío)	Fichero OSM a autoimportar al inicio si la BD está vacía
`SIMULATION_TICK_RATE`	`0.1`	Segundos entre actualizaciones de la simulación
`MAX_VEHICLES`	(sin límite)	Número máximo de vehículos en simulación
`POSTGRES_HOST`	`localhost`	*Host* de PostgreSQL (`postgres` en Docker)
`POSTGRES_PORT`	`5432`	Puerto de PostgreSQL
`POSTGRES_USER`	`dt_user`	Usuario de la base de datos
`POSTGRES_PASSWORD`	`dt_password`	Contraseña de la base de datos
`POSTGRES_DB`	`digital_twin`	Nombre de la base de datos
`DB_POOL_SIZE`	`5`	Tamaño del grupo de conexiones
`DB_MAX_OVERFLOW`	`10`	Conexiones adicionales sobre el grupo

Cuadro B.3: Principales variables de entorno del *backend* (fichero `.env`) y sus valores por defecto

Constante	Por defecto	Descripción
`BACKEND_HOST`	`localhost`	*Host* del *backend*
`BACKEND_PORT`	`8000`	Puerto del *backend*
`BACKEND_PROTOCOL`	`http`	Protocolo de la API
`HTTP_TIMEOUT_SECONDS`	`10.0`	Tiempo máximo de espera de las peticiones HTTP
`NETWORK_PAGE_SIZE`	`1000`	Tamaño de página en la carga paginada de la red

Cuadro B.4: Constantes de conexión del cliente Godot (fichero `config/config.gd`) que con más frecuencia requieren ajuste

B.8 Resolución de problemas frecuentes

El Cuadro B.5 reúne los problemas más habituales en la puesta en marcha y su solución.

Síntoma	Causa probable y solución
El cliente no conecta (*connection refused*)	El *backend* no está levantado o la dirección no coincide. Verificar el *health check* y las constantes de conexión de `config.gd`.
El *backend* no arranca y no conecta con PostgreSQL	La base de datos aún no está lista. Con Docker, la condición de salud lo evita; en instalación manual, arrancar PostgreSQL antes que el *backend*.
Un puerto (8000, 5432 o 5050) está ocupado	Otro proceso lo usa. Cambiar `PORT` en `.env` o el mapeo de puertos en `docker-compose.yml`.
El cliente muestra la red vacía (0 nodos)	No se ha importado el mapa. Ejecutar `load_osm.py` o el extremo `/api/map/import` (§ B.4).
Error al construir las *wheels* en Docker (p. ej. *shapely*) bajo `3.14t`	Faltan herramientas de compilación. La imagen ya instala `gcc`, `g++`, `build-essential` y `libpq-dev`; en instalación manual, instalarlas en el *host*.
El canal WS se cierra con mensajes grandes	La trama supera el límite por defecto. Arrancar *uvicorn* con `--ws-max-size 4194304` (ya incluido en el `Dockerfile`).
pgAdmin solicita una contraseña maestra	La orquestación ya la desactiva por configuración; basta usar las credenciales de `.env`.

Cuadro B.5: Problemas frecuentes en la puesta en marcha del sistema y su solución

Referencias

[AMHH+18] T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, y S. Hillaire. *Real-Time Rendering*. CRC Press, Boca Raton, FL, edición 4.ª, 2018.

[Ayu25] Ayuntamiento de Talavera de la Reina. Ordenanza reguladora de la Zona de Bajas Emisiones. Aprobación inicial 31/10/2025; Boletín Oficial de la Provincia de Toledo, n.º 219, 14/11/2025, inserción 5524, 2025.

[Ban08] D. Banister. The sustainable mobility paradigm. *Transport Policy*, 15(2):73–80, 2008.

[Bat18] M. Batty. Digital twins. *Environment and Planning B: Urban Analytics and City Science*, 45(5):817–820, 2018.

[BBv+01] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, et al. Manifesto for Agile Software Development. `https://agilemanifesto.org/`, 2001.

[BCK21] L. Bass, P. Clements, y R. Kazman. *Software Architecture in Practice*. Addison-Wesley, Boston, edición 4.ª, 2021.

[BDG+16] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, y R. F. Werneck. Route Planning in Transportation Networks. En *Algorithm Engineering: Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, páginas 19–80. Springer, 2016.

[Bea10] D. Beazley. Understanding the Python GIL. PyCON 2010, 2010. Charla técnica, transcripción y diapositivas en `http://www.dabeaz.com/python/UnderstandingGIL.pdf`.

[Ber01] Y. W. Bernier. Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization. Game Developers Conference (GDC). Valve Software. `https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization`, 2001.

[Bra17] T. Bray. RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format. Internet Engineering Task Force, 2017.

[Cas26] Castilla-La Mancha Media. El Tajo, en nivel naranja, en Talavera: hay cortes de tráfico y viviendas inundadas tras la subida del nivel freático. CMM Media, 5 de febrero de 2026. https://www.cmmedia.es/noticias/castilla-la-mancha/toledo/inundaciones-cortes-talavera-tajo-nivel-rojo.html, 2026.

[Caz26] Julián Cazallas. La subida del río Tajo a su paso por Talavera de la Reina provoca inundaciones en garajes de varias calles. El Español — El Digital de Castilla-La Mancha, 4 de febrero de 2026. https://www.elespanol.com/eldigitalcastillalamancha/region/toledo/20260204/subida-rio-tajo-paso-talavera-reina-provoca-inundaciones-garajes-varias-calles/1003744117312_0.html, 2026.

[CB94] D. Clegg y R. Barker. *Case Method Fast-Track: A RAD Approach*. Addison-Wesley, Wokingham, England, 1994.

[Ces24] Cesium GS, Inc. CesiumJS: 3D geospatial visualization for the web. https://cesium.com/platform/cesiumjs/, 2024. Consultado en 2025.

[CFG+10] J. Casas, J. L. Ferrer, D. García, J. Perarnau, y A. Torday. Traffic Simulation with Aimsun. En J. Barceló, editor, *Fundamentals of Traffic Simulation*, páginas 173–232. Springer, New York, 2010.

[CLM26] CLM24. Talavera aplaza su Zona de Bajas Emisiones por defectos de forma. CLM24 — Castilla-La Mancha 24, 27 de marzo de 2026. https://www.clm24.es/articulo/toledo/talavera-aplaza-zona-bajas-emisiones-defectos-forma/20260327181748473894.html, 2026.

[Cor21] Cortes Generales. Ley 7/2021, de 20 de mayo, de cambio climático y transición energética. Boletín Oficial del Estado núm. 121, 2021. BOE-A-2021-8447.

[Cov24] Cover Talavera. La implantación de la ZBE avanza en Talavera con la adjudicación del contrato de obra. Cover Talavera, 2 de enero de 2024. https://www.covertalavera.com/la-implantacion-de-la-zbe-avanza-en-talavera-con-la-adjudicacion-del-contrato-de-obra/, 2024.

[CR74] E. Catmull y R. Rom. A Class of Local Interpolating Splines. En R. E. Barnhill y R. F. Riesenfeld, editors, *Computer Aided Geometric Design*, páginas 317–326. Academic Press, 1974.

[Dag95] C. F. Daganzo. The cell transmission model, part II: Network traffic. *Transportation Research Part B: Methodological*, 29(2):79–93, 1995.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[Doy23] Doymo Ingeniería y Arquitectura. Plan de Movilidad Urbana Sostenible de Talavera de la Reina. Documento técnico encargado por el Ayuntamiento de Talavera de la Reina, 2023.

[DP73] D. H. Douglas y T. K. Peucker. Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[DRC+17] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, y V. Koltun. CARLA: An Open Urban Driving Simulator. En *Proceedings of the 1st Annual Conference on Robot Learning (CoRL)*, páginas 1–16, 2017.

[Dri10] V. Driessen. A successful Git branching model. `https://nvie.com/posts/a-successful-git-branching-model/`, 2010.

[Esp24] La Zona de Bajas Emisiones de Talavera de la Reina se activará, pero sin restricciones ni multas. El Español — El Digital de Castilla-La Mancha, 11 de noviembre de 2024. `https://www.elespanol.com/eldigitalcastillalamancha/region/toledo/20241111/zona-bajas-emisiones-talavera-reina-activara-sin-restricciones-multas/900410278_0.html`, 2024.

[Eva03] E. Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, Boston, 2003.

[Fie00] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[FM11] I. Fette y A. Melnikov. RFC 6455: The WebSocket Protocol. Internet Engineering Task Force, 2011.

[Fow02] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, Boston, 2002.

[FV10] M. Fellendorf y P. Vortisch. Microscopic Traffic Flow Simulator VISSIM. En J. Barceló, editor, *Fundamentals of Traffic Simulation*, páginas 63–93. Springer, New York, 2010.

[GHJV94] E. Gamma, R. Helm, R. Johnson, y J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1994.

[Gip81] P. G. Gipps. A behavioural car-following model for computer simulation. *Transportation Research Part B: Methodological*, 15(2):105–111, 1981.

[Gob03] Gobierno de España. Real Decreto 1428/2003, de 21 de noviembre, por el que se aprueba el Reglamento General de Circulación. Boletín Oficial del Estado núm. 306, 2003. BOE-A-2003-23514.

[Gob20] Gobierno de España. Real Decreto 970/2020, de 10 de noviembre, por el que se modifican el Reglamento General de Circulación y el Reglamento General de Vehículos en materia de medidas urbanas de tráfico. Boletín Oficial del Estado núm. 297, 2020. BOE-A-2020-14048; en vigor desde el 11 de mayo de 2021.

[Gob22] Gobierno de España. Real Decreto 1052/2022, de 27 de diciembre, por el que se regulan las zonas de bajas emisiones. Boletín Oficial del Estado núm. 312, 2022. BOE-A-2022-22389.

[Gri14] M. Grieves. Digital Twin: Manufacturing Excellence through Virtual Factory Replication. Technical report, Florida Institute of Technology, 2014. White Paper basado en la conferencia original de 2003.

[Gro23] S. Gross. PEP 703 – Making the Global Interpreter Lock Optional in CPython. Python Enhancement Proposal, 2023. `https://peps.python.org/pep-0703/`.

[GS12] E. Glaessgen y D. Stargel. The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles. En *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, 2012.

[Gut84] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. En *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, páginas 47–57. ACM, 1984.

[HNA16] A. Horni, K. Nagel, y K. W. Axhausen, editors. *The Multi-Agent Transport Simulation MATSim*. Ubiquity Press, London, 2016.

[HNR68] P. E. Hart, N. J. Nilsson, y B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[HSS08] A. A. Hagberg, D. A. Schult, y P. J. Swart. Exploring Network Structure, Dynamics, and Function using NetworkX. En *Proceedings of the 7th Python in Science Conference (SciPy 2008)*, páginas 11–15, Pasadena, CA, 2008.

[HW08] M. Haklay y P. Weber. OpenStreetMap: User-Generated Street Maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.

[INF19] INFRAS, Bern. *Handbook Emission Factors for Road Transport (HBEFA) 4.1 – Documentation*, 2019.

[KEBB12] D. Krajzewicz, J. Erdmann, M. Behrisch, y L. Bieker. Recent Development and Applications of SUMO – Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, 2012.

[KTH07] A. Kesting, M. Treiber, y D. Helbing. General Lane-Changing Model MOBIL for Car-Following Models. *Transportation Research Record: Journal of the Transportation Research Board*, 1999(1):86–94, 2007.

[LBBW+18] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, y E. Wießner. Microscopic Traffic Simulation using SUMO. En *21st International Conference on Intelligent Transportation Systems (ITSC)*, páginas 2575–2582. IEEE, 2018.

[LW55] M. J. Lighthill y G. B. Whitham. On Kinematic Waves. II. A Theory of Traffic Flow on Long Crowded Roads. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 229(1178):317–345, 1955.

[Min24a] Ministerio de Transportes, Movilidad y Agenda Urbana. Mapa de tráfico de la Red de Carreteras del Estado. `https://mapadetrafico.transportes.gob.es/`, 2024. Plan Nacional de Aforos.

[Min24b] Ministerio de Transportes y Movilidad Sostenible. Transportes adjudica por 7 millones de euros las obras para adecuar al paso de peatones y ciclistas la N-502a, en Talavera de la Reina. Nota de prensa, 3 de octubre de 2024. `https://www.transportes.gob.es/el-ministerio/sala-de-prensa/noticias/jue-03102024-1245`, 2024.

[MRRK+20] N. Mueller, D. Rojas-Rueda, H. Khreis, M. Cirach, D. Andrés, J. Ballester, X. Bartoll, C. Daher, A. Deluca, C. Echave, C. Milà, S. Márquez, J. Palou, K. Pérez, C. Tonne, M. Stevenson, S. Rueda, y M. Nieuwenhuijsen. Changing the urban design of cities for health: The superblock model. *Environment International*, 134:105132, 2020.

[New02] G. F. Newell. A simplified car-following theory: a lower order model. *Transportation Research Part B: Methodological*, 36(3):195–205, 2002.

[NGKS09] L. Ntziachristos, D. Gkatzoflias, C. Kouridis, y Z. Samaras. COPERT: A European Road Transport Emission Inventory Model. En *Information Technologies in Environmental Engineering*, páginas 491–504. Springer, 2009.

[OH21] R. O. Obe y L. S. Hsu. *PostGIS in Action*. Manning, Shelter Island, NY, edición 3.ª, 2021.

[Ope10] Open Geospatial Consortium. *OpenGIS Implementation Standard for Geographic Information — Simple Feature Access — Part 2: SQL Option*, edición 1.2.1, 2010. OGC 06-103r4.

[Ram72] U. Ramer. An Iterative Procedure for the Polygonal Approximation of Plane Curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972.

[Ram22] L. Ramalho. *Fluent Python: Clear, Concise, and Effective Programming*. O'Reilly Media, Sebastopol, CA, edición 2.ª, 2022.

[RTC10] F. Ramm, J. Topf, y S. Chilton. *OpenStreetMap: Using and Enhancing the Free Map of the World*. UIT Cambridge, Cambridge, 2010.

[SGD11] C. Sommer, R. German, y F. Dressler. Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis. *IEEE Transactions on Mobile Computing*, 10(1):3–15, 2011.

[Sny87] J. P. Snyder. *Map Projections: A Working Manual*. Number 1395 in U.S. Geological Survey Professional Paper. U.S. Government Printing Office, Washington, D.C., 1987.

[SPVPRR21] R. Salas, M. J. Perez-Villadoniga, J. Prieto-Rodriguez, y A. Russo. Were traffic restrictions in Madrid effective at reducing NO_2 levels? *Transportation Research Part D: Transport and Environment*, 91:102689, 2021.

[TCQ+18] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, y F. Sui. Digital twin-driven product design, manufacturing and service with big data. *The International Journal of Advanced Manufacturing Technology*, 94:3563–3576, 2018.

[THH00] M. Treiber, A. Hennecke, y D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805–1824, 2000.

[TK13] M. Treiber y A. Kesting. *Traffic Flow Dynamics: Data, Models and Simulation*. Springer, Berlin Heidelberg, 2013.

[U.S20] U.S. Environmental Protection Agency, Washington, DC. *MOVES3: Latest Version of Motor Vehicle Emission Simulator*, 2020. EPA-420-F-20-050.

[Ven22] C. Veness. Calculate distance, bearing and more between latitude/longitude points. Movable Type Scripts. https://www.movable-type.co.uk/scripts/latlong.html, 2022. Consultado en 2026.

[vO24] vis.gl y OpenJS Foundation. deck.gl: WebGL-powered framework for geospatial visualization. https://deck.gl/, 2024. Consultado en 2025.

[YSK11] C. Yuksel, S. Schaefer, y J. Keyser. Parameterization and Applications of Catmull-Rom Curves. *Computer-Aided Design*, 43(7):747–755, 2011.

Este documento fue editado y tipografiado con LaTeX
empleando la clase **gita-tfg** que se puede encontrar en:
`https://github.com/anarubioruiz/gita-tfg`

La clase **gita-tfg** está basada en la clase **esi-tfg**:
`https://github.com/UCLM-ESI/esi-tfg`