EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Maestría

Desarrollo de un espacio unificado de datos de movilidad en ciudades

presentada por

Ing. Luis Angel Reyes González

como requisito para la obtención del grado de
Maestría en Ciencias Computacionales

Director de tesis
Dr. Hugo Estrada Esquivel

Co-directora de tesis
Dra. Alicia Martínez Rebollar

Cuernavaca, Morelos, México. Febrero de 2023.

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

Centro Nacional de Investigación
y Desarrollo Tecnológico
Departamento de Ciencias Computacionales

Cuernavaca, Mor., 03/febrero/2023

OFICIO No. DCC/035/2023
Asunto: Aceptación de documento de tesis
CENIDET-AC-004-M14-OFICIO

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes del Comité Tutorial de LUIS ANGEL REYES GONZÁLEZ, con número de control M21CE024, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis de grado titulado "DESARROLLO DE UN ESPACIO UNIFICADO DE DATOS DE MOVILIDAD EN CIUDADES", y hemos encontrado que se han atendido todas las observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

HUGO ESTRADA ESQUIVEL
Director de tesis

ALICIA MARTÍNEZ REBOLLAR
Codirectora de tesis

JAVIER ORTIZ HERNÁNDEZ
Revisor 1

MARÍA YASMÍN HERNÁNDEZ PÉREZ
Revisor 2

C.c.p. Depto. Servicios Escolares.
Expediente/ Estudiante
JGGS/lbm

SEP TecNM CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
RECIBIDO
07 FEB 2023
SUBDIRECCIÓN ACADÉMICA

EDUCACIÓN
07 FEB 2023
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SERVICIOS ESCOLARES
RECIBIDO
cenidet

Cuernavaca, Mor., 07/febrero/2023
No. De Oficio: SAC/040/2023
Asunto: Autorización de impresión de tesis

LUIS ÁNGEL REYES GONZÁLEZ
CANDIDATO AL GRADO DE MAESTRO EN CIENCIAS DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"DESARROLLO DE UN ESPACIO UNIFICADO DE DATOS DE MOVILIDAD DE CIUDADES"**, ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un cordial saludo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Conocimiento y tecnología al servicio de México"

EDUCACIÓN
CENTRO NACIONAL DE INV... Y DESARROLLO TECNOL...
SUBDIRECCIÓN ACADÉ...

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO

C. c. p. Departamento de Ciencias Computacionales
Departamento de Servicios Escolares

CMAZ/RMA

EDUCACIÓN
08 FEB 2023
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SERVICIOS ESCOLARES
RECIBIDO
EBN

Dedicatoria

A mi novia, futura prometida y esposa Guadalupe Vianey Martínez, por su amor y apoyo incondicional. Eres el combustible de mi vida, Te amo.

A mis padres, Ángela González y Rubén Reyes quienes me han dado lo mejor de ellos de forma incondicional y que han sido para mí un ejemplo a seguir, los amo mucho.

A mis hermanos, Diego Armando Reyes y Julio Cesar Reyes, quienes me han brindado su apoyo y que siempre me han hecho ver el lado amable de las cosas, los amo mucho.

Agradecimientos

A Dios todo poderoso por siembre brindar la fortaleza y las virtudes necesarias para finalizar cualquiera de mis proyectos.

Al Centro Nacional de Investigación y Tecnología (CENIDET) por brindarme la oportunidad de formar parte del alumnado de esta gran institución.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo económico que me otorgó para desarrollar este proyecto de tesis.

A mi director de tesis, Dr. Hugo Estrada Esquivel por sus consejos, observaciones y paciencia para atender mi proyecto de tesis logrando así concluir con los objetivos de mi proyecto.

A mi codirectora de tesis, Dra. Alicia Martínez Rebollar por brindarle las herramientas y mostrarme el camino correcto para obtener buenos resultados dentro de mi proyecto de tesis.

A mis revisores, la Dra. María Yasmín Hernández Pérez y el Dr. Javier Ortiz Hernández por todas sus observaciones y sugerencias que me ayudaron a mejorar mi proyecto de tesis.

A mis compañeros y amigos, quienes hicieron que mi instancia en Cuernavaca y en CENIDET fuera una experiencia inigualable y muy agradable.

¡Les agradezco a todos infinitamente!

Resumen

Hoy en día, las grandes ciudades producen una enorme cantidad de información proveniente de múltiples servicios públicos y privados. Una de las fuentes de datos que ha tenido especial interés para el desarrollo de Ciudades Inteligentes es la movilidad urbana.

La movilidad ha sido uno de los factores que más ha favorecido el crecimiento económico, pero también ha generado múltiples problemas en las ciudades. Por esta razón, la movilidad también ha sido objeto de investigación por parte de muchos grupos a nivel internacional.

Uno de los temas relevantes es el análisis de datos que se generan en tiempo real en algunas grandes ciudades utilizando agentes y bases de datos. Sin embargo, estos datos se producen en formatos muy diferentes y usualmente sin considerar estándares, lo cual dificulta que se puedan usar los datos para desarrollar soluciones de movilidad inteligente.

En este proyecto de tesis se propone un espacio unificado de datos implementado con tecnologías de la Plataforma FIWARE. Este espacio unificado contiene datos en tiempo real y datos históricos de la ubicación de los Metrobuses y la congestión vehicular de la Ciudad de México. Estos datos se encuentran dispersos en diferentes espacios de almacenamiento y son representados con diferentes modelos de datos. Es decir, los datos de localización de Metrobuses y de tráfico se encuentran en diferentes fuentes de datos y no están representados usando estándares, por lo que se proponen prototipos software que se encargan de adquirir los datos de las fuentes originales, transformar los datos a un modelo de datos del estándar NGSI (Interfaces de Servicio de Nueva Generación) y enviar los datos al espacio unificado basado en FIWARE.

Como resultado, los desarrolladores y organizaciones interesadas por datos de movilidad podrán consultar el API (Interfaz de Programación de Aplicaciones) de nuestro espacio unificado de datos y consumir los datos de movilidad ya estandarizados. Estos datos permitirán generar nuevo conocimiento o alguna aplicación que satisfaga necesidades concretas de movilidad.

Abstract

Currently, the big cities produce an enormous amount of information coming from several public and private services. Urban mobility is one of the relevant data sources in the construction of Smart Cities.

The mobility is one of the factors that had better promote the economic growth, but it has also generated multiple issues in cities. For this reason, mobility has also been the subject of research by many groups internationally.

One of the relevant topics is the analysis of data that is generated in real time in some large cities using agents and databases. However, these data are produced in very different formats and usually without considering standards, which makes it difficult to use the data to develop smart mobility solutions.

In this thesis project, a unified data space implemented with FIWARE Platform technologies is proposed. This unified space contains real-time data and historical data on the location of Metrobuses and traffic congestion in Mexico City. These data are scattered in different storage spaces and are represented with different data models. That is, the location data of Metrobuses and traffic are found in different data sources and are not represented using standards, so software prototypes are proposed that are responsible for acquiring the data from the original sources, transforming the data into a data model of the NGSI (New Generation Service Interfaces) standard and send the data to the unified space based on FIWARE.

As a result, developers and organizations interested in mobility data will be able to query the API (Application Programming Interface) of our unified data space and consume the already standardized mobility data. These data will make it possible to generate new knowledge or an application that satisfies specific mobility needs.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1
Introducción

1.1 Introducción

En la actualidad, las grandes ciudades son consideradas como las principales fuentes económicas de cualquier país. Esta consideración está basada en el hecho que en las grandes ciudades se pueden encontrar el mayor número de comercios, aglomeraciones empresariales y servicios, tales como, transporte, tecnología, comunicación vial, telecomunicaciones, etcétera [1]. La movilidad en ciudades es uno de los factores que más ha favorecido el crecimiento económico [2]. Sin embargo, esta misma movilidad se ha convertido en uno de los problemas principales para los habitantes de las ciudades debido a los problemas que se generan al no contar con un buen sistema de predicción de tráfico para prevenir aglomeraciones o inclusive hacer un mejor uso del transporte público [3].

Alguno nuevos sistemas de gestión urbana impulsada por enfoques como el Internet de las Cosas (IoT por sus siglas en ingles *Internet of Things*) proveen gran cantidad de datos acerca de la movilidad vehicular en una ciudad. Sin embargo, uno de los grandes problemas de estos sistemas es que la información se encuentra aislada en múltiples espacios de almacenamiento, y bajo diferentes modelos de representación [4] [5] [6] [7]. Estos modelos de representación no toman en cuenta el uso de estándares que permitan la reutilización de los datos almacenados.

Hoy en día, por ejemplo, algunos sistemas de transporte público en grandes ciudades (ej. Ciudad de México) proveen información de la localización en tiempo real de algunas de sus unidades de transporte, al igual que existen plataformas web que brindan información sobre la congestión vehicular de la ciudad también en tiempo real [8] [7]. Sin embargo, en ambos casos la información no se encuentra estandarizada y el acceso a los datos es muy complicado. Por ejemplo, en el caso de la información de tráfico obtenida de la Plataforma HERE Maps [7], la información de toda la Ciudad de México se encuentra dividida en miles de segmentos, lo cual a su vez contienen información de los miles de calles y avenidas de esta ciudad. El consumo directo de esta información constituye un reto para los desarrolladores que deseen consumir estos datos para tareas de análisis de información.

Actualmente se cuenta con diferentes plataformas especializadas en el IoT que cuentan con estándares y protocolos para el manejo y procesamiento de información a gran escala, con el objetivo de crear y desplegar aplicaciones inteligentes y administrables [9] [10] [11]. Una de las plataformas especializada en el IoT es FIWARE [6]. Esta plataforma del IoT brinda la posibilidad de construir aplicaciones de movilidad inteligente que almacenen sus datos en la nube. Sin embargo, es necesario construir mecanismos que permitan el acceso a los datos, su estandarización y almacenamiento en una nube que permita la reutilización de los datos por desarrolladores o analistas externos.

Este trabajo de investigación tiene como objetivo presentar una plataforma de software para la construcción del espacio unificado de datos para movilidad en ciudades dentro de la Plataforma FIWARE. El enfoque propuesto en esta tesis se encarga de adquirir datos de transporte público y congestión vehicular en tiempo real y estandarizarlos en la Plataforma FIWARE [12] para su uso en aplicaciones que satisfagan alguna necesidad de movilidad urbana.

En forma más concreta, en este trabajo de investigación se utilizan los datos provistos por el Sistema de Corredores de Transporte Público de Pasajeros de la Ciudad de México, conocido como

Metrobús, y los datos sobre la congestión vehicular de la Ciudad de México compartidos por la Plataforma HERE Maps.

Nuestro enfoque permite la estandarización de los datos de ambas fuentes utilizando modelos de datos de la Plataforma FIWARE. Los datos estandarizados son enviados a un servidor FIWARE, de tal forma que puedan ser consultados por desarrolladores utilizando servicios REST. Los desarrolladores solo requieren conocer la estructura de los modelos de datos FIWARE para realizar consultas al espacio unificado.

1.2 Planteamiento del problema

En algunas ciudades del mundo, los tiempos utilizados en viajes son generalmente altos. El crecimiento demográfico acelerado, la concentración de las fuentes de trabajo en las urbes y una inexistente planificación urbana, han provocado que las ciudades se encuentren con problemas de movilidad [13]. Esta movilidad disminuye aún más para algunos usuarios que utilizan el servicio del transporte público debido a un servicio ineficiente y a la congestión vehicular.

Una de las principales causas de problemas de movilidad de los usuarios de transporte público es que las rutas por donde transitan generalmente coinciden con las intersecciones con el flujo vehicular más alto [14]. Además de que, las redes de transporte público en su gran mayoría siguen una ruta establecida y no permiten evitar la congestión vehicular al cruzar por alguna zona con tráfico [14].

Uno de los enfoques que está teniendo impacto en la solución de problemas de las ciudades es el Internet de las Cosas. Diariamente se observan una cantidad enorme de dispositivos IoT que cuentan con la tecnología requerida para compartir datos de sí mismos o de sus entornos.

Sin embargo, una de las problemáticas que se tienen en los proyectos donde se utilizan dispositivos IoT para compartir datos de cualquier tipo es la falta de implementación de estándares. La creación de soluciones IoT sin la implementación de estándares incrementa considerablemente el costo del proyecto, además de poner en riesgo su adopción a gran escala y frena la innovación tecnológica de iniciativas de proyectos para Ciudades Inteligentes.

Si bien existen ciudades que presumen haber implementado proyectos con tecnologías IoT, la ausencia en la estandarización de los datos hace que una solución que funciona en una ciudad, no pueda funcionar en otra sin esfuerzos de adaptación muy importantes. Esto se debe a que la ausencia de estándares para la representación de datos de ciudades dificulta en extremo su consumo por desarrolladores externos que deseen utilizar estos datos para crear soluciones. Esto hace que el esfuerzo de publicar los datos en formatos abiertos no tenga una recompensa clara en su uso a gran escala.

Para atender estos inconvenientes, en esta tesis se propone el desarrollo de un espacio unificado de datos, prototipos y métodos funcionales que aporten a la estandarización de datos de movilidad urbana. Este espacio unificado será implementado para su uso en aplicaciones que satisfagan alguna necesidad de movilidad urbana en la Plataforma FIWARE.

Las soluciones que ofrece la Plataforma FIWARE pretenden impulsar estándares para recopilar, gestionar y publicar información de contexto y adicionalmente aportar elementos que permitan explotar la información una vez recopilada.

Una estandarización como la que brinda la Plataforma FIWARE resulta clave para contribuir al desarrollo de un mercado digital para las aplicaciones inteligentes donde las apps y soluciones puedan trasladarse de un cliente a otro sin muchos cambios.

En este sentido, el espacio unificado de datos y los prototipos desarrollados en este proyecto de tesis tienen la finalidad de contribuir a la disminución de la complejidad de los datos generados por sensores y traducirlos en un lenguaje común estandarizando para su representación y publicación.

En este proyecto de tesis se propone extraer datos de dos diferentes fuentes de información: Metrobuses y congestión vehicular y utilizar los modelos de datos *Vehicle* y *TrafficFlowObserved* de FIWARE para estandarizar estos datos y facilitar su consulta.

Los datos estandarizados serán alojados en la aplicación Quantumleap. Esta herramienta forma parte de las soluciones IoT con la que cuenta la Plataforma FIWARE. Esta solución ayudará al consumo y análisis de los datos de movilidad urbana para generar aplicaciones inteligentes o conocimiento que ayuden a la mejora del servicio de transporte público.

1.3 Solución propuesta

Los datos generados por los diferentes dispositivos IoT de una ciudad son alojados en distintos espacios de almacenamiento bajo modelos de representación no estandarizados, provocando que los datos sean difíciles de extraer y comprender.

En este proyecto de tesis se propone el desarrollo de componentes software para consumir dos diferentes fuentes de datos de movilidad urbana de la Ciudad de México: (1) Datos de la ubicación en tiempo real de los Metrobuses de la Ciudad de México. (2) Datos de la congestión vehicular de la Ciudad de México. Ambas fuentes de datos no utilizan modelos de representación estandarizados.

Las fuentes de datos de los Metrobuses y de la congestión vehicular de la CDMX no utilizan modelos de representación de datos estandarizados, por lo que se propone un método de estandarización de los datos de las dos diferentes fuentes. La Plataforma FIWARE brinda una serie de modelos de representación de datos estandarizados de fácil comprensión. Para la representación de los datos de la ubicación de los Metrobuses de la Ciudad de México se utilizó el modelo de datos *Vehicle*, mientras que, para la representación de los datos de la congestión vehicular de las Ciudad de México se utilizó el modelo de datos *TrafficFlowObserved*. Ambos modelos de datos son del estándar NGSI y cuentan con los atributos necesarios y genéricos que permiten representar la ubicación de un vehículo de transporte público y la congestión vehicular de un segmento de la Ciudad de México.

Adicionalmente, la forma de extraer los datos de la dos diferentes fuentes son totalmente distintas y cada una tiene un grado de complejidad. En este proyecto de tesis se propone alojar los datos estandarizados en un servidor con las funcionalidades que ofrece la aplicación Quantumleap [21] de la Plataforma FIWARE. Esta solución permite que desde una sola API se pueda tener acceso a datos de la ubicación de los Metrobuses y la congestión vehicular de la Ciudad de México. Asimismo,

se podrá tener acceso a datos en tiempo real y datos históricos, lo que permite realizar análisis estadísticos para la toma de decisiones o alguna aplicación que beneficie a la movilidad urbana.

En la Figura 1.1 muestra la metodología de solución propuesta, la cual está conformada por 5 procesos principales: (1) Análisis de fuentes de las datos. (2) Análisis de los componentes de FIWARE y sus estructuras de datos. (3) Creación de las reglas de mapeo de datos. (4) Desarrollo del espacio unificado de datos. (5) Pruebas y validación del prototipo.

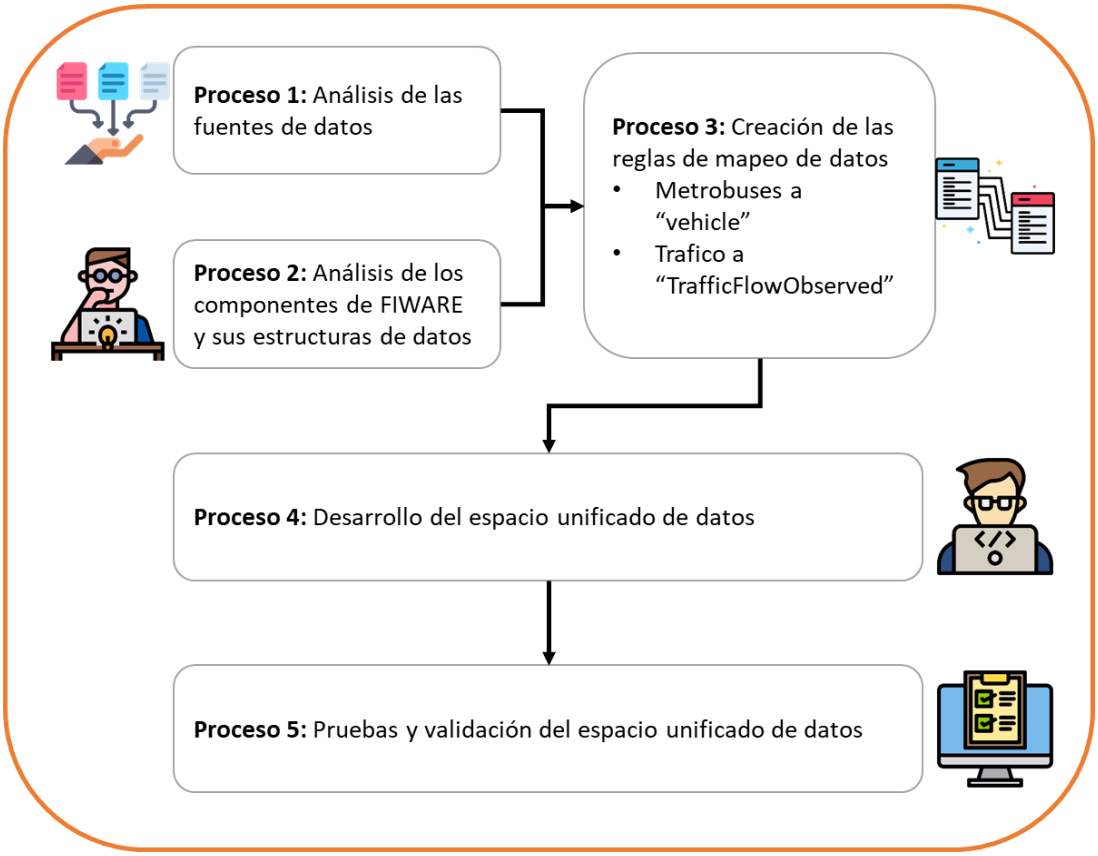


Figura 1.1 Metodología de solución

Se brinda una breve descripción de las actividades que se realizan en cada una de los procesos de la metodología de solución:

Proceso 1. Análisis de las fuentes de datos. En este proceso se estudiaron las dos fuentes de datos que se utilizan en este proyecto de tesis: (1) Plataforma con datos abiertos de los Metrobuses de la Ciudad de México [8]. (2) HEREMaps con los datos de congestión vehicular de la Ciudad de México [27]. De ambas fuentes de datos se examinaron los siguientes puntos: (1) Procedimiento para la extracción de los datos. (2) Comprensión de la estructura de datos. (3) Método para recorrer cada una de las instancias encontradas. (4) Método para acceder al valor de atributos específicos.

Proceso 2. Análisis de los componentes de FIWARE y sus estructuras de datos. En este proceso se llevó a cabo un estudio sobre el funcionamiento de los componentes de la Plataforma FIWARE, asimismo se analizaron los estándares de los modelos de datos que utiliza la Plataforma FIWARE y se identificaron los modelos que mejor se adaptaron a las necesidades de este proyecto de tesis. De

igual forma se investigaron los métodos necesarios para la implementación de un servidor local con los componentes de la Plataforma FIWARE para su uso a lo largo del desarrollo de este proyecto de tesis.

Proceso 3. Creación de las reglas de mapeo de datos. En este proceso se crearon las reglas de mapeo entre los modelos de datos: (1) Mapeo de los datos de los Metrobuses al modelo *Vehicle* del estándar NGSI. (2) Mapeo de los datos de congestión vehicular al modelo de datos *TrafficFlowObserved* del estándar NGSI. Al mismo tiempo se diseñaron los métodos para realizar transformaciones equivalentes de los valores de los atributos que lo requieran.

Proceso 4. Desarrollo del espacio unificado de datos. En este proceso se implementó un servidor local con las funcionalidades de los componentes de FIWARE. Este servidor se encuentra localizado en el SITE de CENIDET. Al mismo tiempo se desarrollaron dos componentes software que se encargan de consumir, mapear y enviar los datos al servidor local de FIWARE. Se desarrollaron dos alimentadores independientes de Datos de Movilidad Urbana (ADMU): (1) ADMU que trabaja con los datos de los Metrobuses el cual se ejecuta cada 65 segundos. (2) ADMU que trabaja con los datos de congestión vehicular el cual se ejecuta cada 5.5 minutos. Cada uno de los mecanismos tiene sus propias reglas de mapeo y sus métodos de transformación de datos.

Proceso 5. Pruebas y validación del espacio unificado de datos. En este proceso se desarrollaron una serie de aplicaciones web demo las cuales consumen datos en tiempo real o datos históricos de nuestro servidor de FIWARE. Los datos consumidos son representados en un mapa donde se aprecia la ubicación de los Metrobuses y la congestión vehicular de la Ciudad de México. De igual manera se desarrollaron una serie de consultas tipo SQL en el componente Grafana para estudiar de forma gráfica el comportamiento de los datos de movilidad de la Ciudad de México. Los datos históricos fueron analizados estadísticamente para la evaluación completa del espacio unificado.

Como resultado de la metodología de solución mencionadas se crea un espacio unificado de datos que contiene datos en tiempo real y datos históricos de los Metrobuses y congestión vehicular de la Ciudad de México.

1.4 Objetivos

1.4.1 Objetivo general

Construir un espacio unificado de datos de movilidad que contenga datos en tiempo real y datos históricos de la ubicación de los Metrobuses de la CDMX y datos de movilidad de la congestión vehicular provenientes de la Plataforma HERE Maps para facilitar el acceso a datos estandarizados por parte de desarrolladores de aplicaciones de movilidad inteligente.

1.4.2 Objetivos específicos

Los objetivos específicos de este proyecto de investigación son:

- Desarrollar una aplicación que permita la obtención de datos, en tiempo real, de Metrobuses y de congestión vehicular a partir de sus fuentes originales.
- Estandarizar los datos de la ubicación de los Metrobuses y de la congestión vehicular utilizando modelos de datos de la Plataforma FIWARE para su almacenamiento y exposición en el espacio unificado.
- Desarrollar una aplicación para el envío de datos estandarizados al componente *Orion Context Broker* y el componente Quantumleap de FIWARE para almacenar, tanto los últimos datos obtenidos en tiempo real, como los datos históricos de los Metrobuses y de la congestión vehicular.
- Realizar pruebas al espacio unificado de datos, visualizando su contenido a través de tableros de control del componente Grafana de FIWARE, así como en mapas para observar el comportamiento de los datos de movilidad.

1.5 Estructura del documento

El contenido de la tesis se encuentra organizada en capítulos de la siguiente manera:

- **Capítulo 2. Marco conceptual:** en este capítulo se definen los principales conceptos que se utilizan a lo largo del desarrollo de este proyecto de tesis.
- **Capítulo 3. Estado del arte:** en este capítulo se describen algunos trabajos de investigación relacionados con el análisis de movilidad urbana en ciudades.
- **Capítulo 4. Metodología de solución para el desarrollo del espacio unificado de datos:** en este capítulo se describe la metodología de la solución propuesta para el desarrollo del espacio unificado de datos de movilidad urbana. Asimismo, se presenta el análisis, diseño y desarrollo de cada uno de los módulos de los prototipos de software propuestos en esta metodología.
- **Capítulo 5: Aplicaciones desarrolladas para el consumo de datos del espacio unificado:** en este capítulo se presenta el desarrollo de unas aplicaciones basas en mapas para la representación de los datos de la ubicación de los metrobuses y congestión vehicular de la CDMX. Asimismo, se desarrollan consultas en el componente Grafana para el análisis del comportamiento de los datos de las entidades guardadas en la Plataforma FIWARE.
- **Capítulo 6: Pruebas y resultados:** en este capítulo se presentan las pruebas realizadas al espacio unificado de datos para validar la correcta estandarización y almacenamiento de los datos.
- **Capítulo 7: Conclusiones y trabajo futuro:** en esta sección se presentan las conclusiones y contribuciones. Asimismo, se presentan los trabajos futuros que se pueden realizar a partir de este proyecto de tesis.

Capítulo 2
Marco conceptual

Las plataformas del Internet de las Cosas han sido implementadas en muchas ciudades como una herramienta para monitorear, controlar y compartir información de servicios públicos como es el caso de la iluminación, recolección de basura, administración de agua, etc. Para lograr esto, las ciudades han implementados redes de sensores que permiten capturar datos del contexto, así como redes de actuadores que permiten la modificación de ese contexto. En este capítulo se describen algunos conceptos utilizados en la implementación de soluciones basadas en plataformas IoT y que son utilizadas en este proyecto de investigación.

2.1 FIWARE y el Internet de las Cosas

A continuación, se presenta los conceptos relevantes para Internet de las Cosas y la Plataforma FIWARE.

2.1.1 Internet de las cosas y Ciudades inteligentes

En la actualidad, el concepto de Internet de las Cosas (IoT por sus siglas en inglés *Internet of Things*) ha tenido gran impacto, debido a que ha permitido conectar dispositivos convencionales al Internet. Estos dispositivos utilizan protocolos de comunicación estándar e interfaces inteligentes con los cuales se integran sin dificultad a la red de comunicación, a su vez, los dispositivos pueden compartir datos asociados con los usuarios y sus entornos. De esta manera, los objetos inteligentes que intervienen dentro de la arquitectura IoT pueden producir información de contexto en grandes cantidades la cual se guarda en la nube [15].

Con el aumento del número de objetos conectados a Internet de las Cosas, los datos generados por estos dispositivos se han incrementado en forma considerable, lo cual ha abierto la puerta al desarrollo de aplicaciones de software que hacen uso de esos datos para análisis de información. Sin embargo, en la mayoría de las ocasiones, los datos que producen estos objetos se encuentran aislados en múltiples espacios de almacenamiento bajo diferentes modelos de representación, siendo así, un gran problema para poder tratar esta información y poder generar nuevo conocimiento o una aplicación que satisfaga alguna necesidad urbana.

El aumento del número de objetos conectados al IoT y a la creciente complejidad de las soluciones obligaron al desarrollo de soluciones para administrar todas las interacciones que existen entre los dispositivos inteligentes y los sistemas software. Estas soluciones deben tener la capacidad de administrar la extensa cantidad de datos que los dispositivos IoT producen. Algunas de las soluciones son: Google Cloud, Amazon Web Service, FIWARE, entre otros [16].

Todos estos dispositivos y plataformas tienen el fin de mejorar la vida cotidiana, la eficiencia del empleo, seguridad, educación y además poder tener un mejor desempeño de los servicios públicos. De esta manera se contempla apoyar las necesidades de las comunidades presentes y futuras en los aspectos económicos, sociales, medioambientales y culturales.

Podemos establecer que una Ciudad Inteligente es aquella que utiliza las Tecnologías de la Información y Comunicaciones (TIC´s) así como los dispositivos y plataformas IoT que se encuentren a su alcance. Está estará centrada no solo en proporcionar servicios más eficientes, sino en transformar las ciudades en plataformas digitales que hagan posible el desarrollo de servicios

abiertos e innovadores para todos los ciudadanos, aportando lo necesario para un nuevo impulso económico, apoyando el crecimiento y la creación de empleos [17].

2.1.2 Plataforma de internet de las cosas FIWARE

FIWARE es una plataforma de código abierto que impulsa la creación de estándares para el desarrollo de aplicaciones y servicios inteligentes para el Internet del Futuro en diferentes dominios [18]. Esta plataforma proporciona un conjunto de Componente Genéricos (GE por sus siglas en inglés *Generic Enablers*) e interfaces de programación de aplicaciones (APIs, por sus siglas en inglés *Application Programming Interface*). Las APIs de la Plataforma FIWARE son públicas y gratuitas, con el propósito de facilitar el desarrollo y despliegue de aplicaciones inteligentes en múltiples sectores y escenarios [19] [12].

2.1.3 Información de contexto

La información de contexto es cualquier información que puede ser usada para caracterizar alguna situación de una entidad. Esta entidad puede ser una persona, algún lugar o cualquier otro objeto de la vida real. Los objetos son considerados relevantes cuando existe interacción entre un usuario y una aplicación.

La cantidad de información que se puede categorizar como información de contexto es inmensa. La información de contexto más común son la ubicación, el tiempo, temperatura, humedad, presión o la actividad de una persona [20]. Para el caso del proyecto de tesis, los datos de contexto para un Metrobús son, por ejemplo, velocidad y localización. Para el caso de la congestión vehicular, los datos de contexto son ocupación de la calle o velocidad promedio.

2.1.4 Componente *Orion Context Broker* de FIWARE

El principal y único componente obligatorio de la solución FIWARE es el *Orion Context Broker* (OCB), el cual permite gestionar el ciclo de vida de la información de contexto, incluyendo actualizaciones, consultar, registros y suscripciones (Figura 2.1). El OCB está rodeado por diversas fuentes de datos (dispositivos IoT, robots y sitios web) que permiten suministrar de datos de contexto, de una forma transparente y sencilla, ocultando la complejidad de la gestión para la captura de datos. El componente Orion Context Borker solo mantiene la información de contexto más actual recibida, sin embargo, para mayor almacenamiento del historial de contexto existen otros componentes, de los cuales en el desarrollo de este trabajo de investigación se utiliza Quantumleap [12].

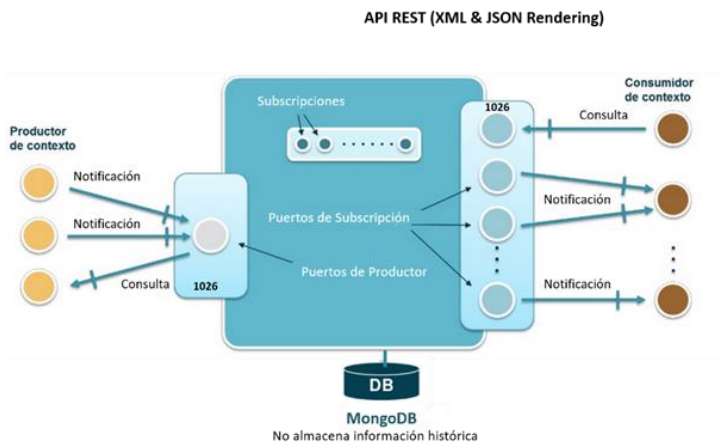


Figura 2.1 *Orion Context Broker*

2.1.5 Componente Quantumleap de FIWARE

El componente Quantumleap es un servicio REST que permite almacenar, consultar y recuperar datos históricos con estándar NGSI, estos datos históricos son generados a partir de los datos en tiempo real que fueron almacenados en el OCB [21]. El componente Quantumleap cuenta con un traductor CrateDB. El trabajo de CrateDB consiste en convertir datos semiestructurado NGSI en forma tabular y los almacena en una base de datos de series temporales, asociando cada registro de base de datos con un índice de tiempo (Figura 2.2). El traductor de CrateDB brinda las ventajas [21]:

- Escalabilidad fácil con clúster de base de datos en contenedores.
- Soporte de Geo-consultas.
- Lenguaje de consulta tipo SQL.
- Integración soportada con herramientas de visualización como Grafana.

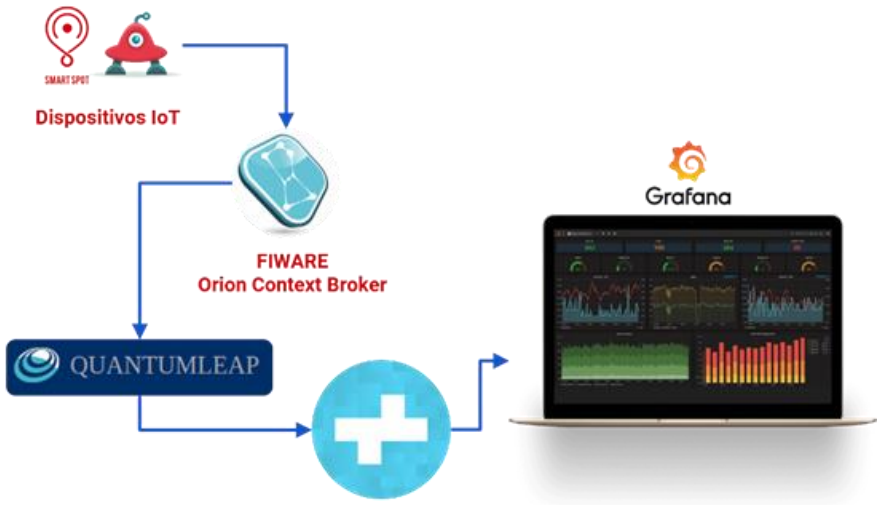


Figura 2.2 Esquema de datos Quantumleap

2.1.6 Estándar NGSI de la Plataforma FIWARE

Uno de los principales propósitos de este proyecto de investigación es la unificación de datos bajo un modelo de representación de FIWARE, estos modelos tienen el estándar NGSI. Los datos deben seguir el estándar NGSI para que puedan ser enviados al servicio *Orion Context Broker*, de esta manera los datos son considerados entidades de contexto.

Una entidad de contexto representa una "cosa", es decir, cualquier objeto físico o lógico. Cada entidad tiene un identificador y un tipo. Los tipos de entidad tienen la intención de describir el tipo de "cosa" que representa esa entidad. En FIWARE todas las entidades se identifican de forma única por la combinación de su identificación y tipo [22] [19].

Los atributos son propiedades de las entidades de contexto. En el modelo de datos NGSI, los atributos tienen un nombre, un tipo, un valor y metadatos [19]. La Figura 2.3 presenta el esquema general de los elementos de contexto de FIWARE, conteniendo un elemento de contexto, sus atributos y sus metadatos. En forma más específica, los modelos se componen de los siguientes elementos:

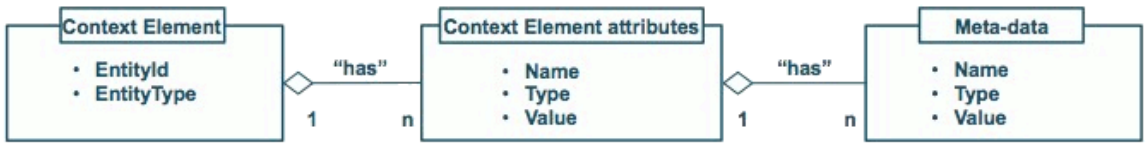


Figura 2.3 Elementos de contexto

- Un EntityId y un EntityType único que identifica la entidad a la cual los datos de contexto hacen referencia.
- Una secuencia de uno o más atributos de elementos de datos (nombre, tipo de dato y valor obtenido).
- Metadatos opcionales vinculados a atributos (nombre, tipo de dato y valor obtenido).

Puesto que en este trabajo de investigación se utilizan los datos de los Metrobuses y congestión de la Ciudad de México, los modelos de datos apropiados para el desarrollo del espacio unificado de datos son:

- *Vehicle*: Representa un vehículo con todas sus características [23].
- *TrafficFlowObserve*: Representa una observación sobre el flujo de tráfico [24].

2.1.7 Notación de Objetos *JavaScript*

La Notación de Objetos de *JavaScript* (JSON por sus siglas en inglés, *JavaScript Objetc Notation*) es un formato ligero de intercambio de datos. Leerlos y escribirlos es simple para el desarrollador, mientras que para las aplicaciones computaciones es simple generarlos e interpretarlos. JSON es una forma de texto que es completamente independiente del lenguaje, pero utiliza reglas que son conocidas por los desarrolladores para poder tratar los datos contenidos [20]. FIWARE utiliza JSON como notación, ya que brinda facilidades para el intercambio de datos.

2.2 Movilidad urbana

En este proyecto de tesis se utilizan los datos de movilidad urbana. La movilidad urbana se define como el conjunto de desplazamientos de un punto a otro de cada uno de los habitantes de una ciudad. Este desplazamiento puede o no utilizar vehículos automotores.

A continuación, se presentan los conceptos relevantes para la movilidad urbana.

2.2.1 Estándar GTFS

La Especificación General de Feeds de Transporte Público (GTFS por sus siglas en inglés *General Transit Feeds Specification*) define un estándar para los horarios de transporte público, y la información geográfica relacionada. Los *feeds GTFS* permiten que las organizaciones de transporte público publiquen sus datos, y que las empresas desarrolladoras implementen aplicaciones que consuman esos datos de manera interoperable [25].

Dentro del ámbito del estándar GTFS existen dos tipos de datos: GTFS estático y GTFS en tiempo real (GTFS-RT por sus siglas en ingles *General Transit Feed Specification – Real Time*). Para el desarrollo de este trabajo de investigación se contempla utilizar los datos GTFS-RT. Estos datos se actualizan cada 60 segundos.

Los *feeds GTFS* en tiempo real proporcionan tres tipos de información actualizada en tiempo real: (1) La posición vehicular con datos de eventos ya producidos. (2) Actualización de viaje con información sobre eventos que ocurran. (3) Un servicio de alertas con información de eventos ocurridos y como afectan los viajes. Los datos de interés para el espacio unificado son las posiciones vehiculares con todos los eventos ya producidos.

2.2.2 Plataforma HERE Maps

El objetivo principal de HEREMaps es brindar una representación digital basada en el flujo automovilístico, con la finalidad de mejorar radicalmente el monitoreo de la congestión vehicular en tiempo real y en puntos geográficos específicos [26].

La Plataforma HEREMaps genera de forma única contenido de ubicación y software que permite a las personas, las empresas y los gobiernos tener acceso a información de este tipo [26]. Uno de los servicios que ofrece esta plataforma es la de brindar APIs de acceso a datos sobre congestión vehicular [27]. Sin embargo, esta información no sigue un estándar genérico y los datos obtenidos son complicados de entender. La información de las APIs de esta plataforma utiliza notación de objetos JSON y XML.

2.2.3 Congestión vehicular

La congestión vehicular se define como el flujo automovilístico de diferentes calles, avenidas, carreteras, etc. El exceso de congestión vehicular afecta a la red vial de toda una zona, generalmente a las zonas metropolitanas puesto que son zonas donde existen mayor incremento demográfico, falta de obras viales alternas, señalizaciones viales desactualizadas, accidentes automovilísticos y aglomeraciones empresarias y de servicios [28].

Las consecuencias del congestionamiento vehicular asociadas con la reducción de velocidad de transito de los vehículos son el retraso en los tiempos de viajes, así como, el incremento del consumo de combustible, desgaste de las obras viales, mayor contaminación ambiental, afectan la calidad de vida y salud de la sociedad que habitan en alguna ciudad metropolitana [28].

Capítulo 3

Estado del arte

En este capítulo se presentan trabajos de investigación relevantes para la temática del procesamiento de datos de movilidad. Estos trabajos relacionados se encuentran clasificados en dos categorías: (1) El análisis de datos de la movilidad en ciudades. (2) Creación y uso de conjuntos de datos de movilidad. Cada uno de los trabajos del estado del arte se presentad describiendo la problemática detectada, la solución propuesta, las tecnologías y técnicas utilizadas para el cumplimiento de los objetivos y las conclusiones encontradas en cada uno de los trabajos.

3.1 Análisis de datos de la movilidad en ciudades

1. Aplicación móvil SedVis para la prevención del sedentarismo

Este trabajo de investigación [29] menciona haber detectado el comportamiento de sedentarismo prolongado como una problemática. Los autores indican que el sedentarismo está relacionado con una serie de riesgos de enfermedades crónicas. Dada a esta alta prevalencia del comportamiento sedentario en la vida diaria, los autores señalan que necesitan soluciones simples pero eficientes para prevenir el comportamiento sedentario de la sociedad.

Una hipótesis del artículo resalta que las visualizaciones basadas en mapas con la ubicación en tiempo real pueden ser eficientes para el usuario del mapa tome conciencia sobre su comportamiento sedentario, ya que proporcionan información sobre el punto geográfico donde se está llevando a cabo la actividad.

El equipo de investigación de este artículo desarrolló una aplicación móvil para teléfonos inteligentes Android denominada “SedVis”. Los autores indican que recopilaron datos de sensores de los teléfonos inteligentes y datos que generalmente comparten los usuarios a la nube de datos de Google. Los datos recopilados por este trabajo de investigación son: (1) Google Activity Recognition API: datos de actividad física. (2) Google Maps API: datos de geolocalización. (3) Google Fit API: datos de actividad deportiva.

La aplicación proporciona visualización de patrones de movilidad personal (tanto para la actividad física como el comportamiento sedentario) y planificaciones de acciones para el cambio de comportamiento sedentario.

El objetivo principal de los autores es investigar el efecto de la visualización del patrón de movilidad en la planificación de acciones de los usuarios para combatir el comportamiento sedentario. El objetivo secundario es evaluar el compromiso del usuario con la visualización y experiencia con la aplicación.

Los autores realizaron un estudio de 3 semanas con 16 participantes que tenían la motivación para reducir su comportamiento sedentario. Dicho comportamiento se estimó en función de los datos de los sensores de sus teléfonos inteligentes, y la aplicación también registró sus planes de acción y la interacción con la aplicación.

La intervención que involucró visualizaciones y planificación de acciones en SedVis tuvo un efecto positivo en la reducción de las horas de sedentarismo de los participantes, con evidencia débil según las estadísticas bayesianas.

En conclusión, el uso de una aplicación de teléfono inteligente para recopilar datos sobre los patrones de movilidad de los usuarios y proporcionar comentarios en tiempo real mediante visualizaciones, puede ser un método prometedor para inducir cambios en el comportamiento sedentario y puede ser más eficaz que la planificación de acciones por sí sola. Los autores mencionan que necesitan replicar su experimentación con muestras más grandes para confirmar estos hallazgos.

2. Visualización de la movilidad urbana de la Ciudad de Sao Paulo

En este trabajo de investigación [30] identifican como problemática la movilidad urbana, ya que afecta directamente a la calidad de vida de la sociedad y tiene un alto impacto en la economía. Este trabajo menciona que el tráfico de la zona metropolitana de Sao Paulo afecta al 89% de la población que necesita trasladarse hacia su trabajo, provocando altas perdidas monetarias.

Los autores indican que los datos de movilidad urbana pueden respaldar la toma de decisiones para contrarrestar las problemáticas relacionadas con la movilidad. Los datos pueden provenir de varias fuentes, tales como: rastreo por GPS, cámara de tráfico, censos y encuestas de viajes. El caso de estudio presentado en el artículo es la Compañía Metropolitana de Sao Paulo, la cual administra el sistema de metro de la ciudad. Esta compañía implementó un estudio de viajes denominado Encuestas Origen-Destino (OD). Esta encuesta recopila datos acerca del trayecto de viajes que un ciudadano puede tener día a día.

Los datos de la encuesta OD contiene puntos de origen y destino de los viajes realizados por una muestra representativa de la población, lo que dio como resultado un aproximado de 42 millones de viajes representados durante 24 horas de un día laboral normal. Los autores consideran que es un gran desafío analizar la extensa cantidad de información que producen todos los días y traducir los datos en imágenes representativas.

Un punto importante que mencionan los autores es que las hojas de cálculo y los programas de estadística ayudan a producir tablas y gráficas, mientras que los mapas de densidad pueden ayudar a mostrar la geolocalización. En este trabajo se propone el uso de un modelo de agrupación con los datos obtenidos de la encuesta OD, de tal forma que se pueda visualizar la estructura y las características de la movilidad urbana en la ciudad de Sao Paulo. La visualización y el análisis de los datos se realizan con ayuda de la herramienta CUBu. Los autores comentan que esta herramienta ha demostrado ser útil en otros escenarios que utilizan datos de movilidad.

Los autores muestran como las visualizaciones agrupadas ayudan a encontrar patrones estructurales en la movilidad urbana y revela nuevos conocimientos sobre el sistema de transporte público y la infraestructura de la ciudad de Sao Paulo que benefician a la movilidad de la ciudad.

En conclusión, este trabajo propone diversas visualizaciones de datos de movilidad urbana de la zona metropolitana de Sao Paulo utilizando métodos de agrupación. Estas visualizaciones ayudan de forma eficiente al análisis y a la toma de decisiones que involucre a la movilidad en la zona estudiada.

3. Visualización de la movilidad del sistema de transporte público

En este trabajo de investigación [31] establece que el sistema de transporte público (PTS) juega un papel importante en las ciudades modernas proporcionando servicios de transporte compartido y masivo. Según los autores, los PTS impactan en los aspectos económicos, ya que podrían reducir el costo total del transporte de la ciudad. Así mismo los autores consideran que un buen transporte público puede asegurar que todos los miembros de la ciudad pueden viajar en menor tiempo, con menor costo y cuidando el medioambiental, ya que el PTS generalmente ahorra más energía que el transporte privado.

El sistema de transporte público, desde la perspectiva de los autores, no solo brinda servicios de movilidad, sino que también comparten datos de los pasajeros, unidades o sus entornos. Sin embargo, el desarrollo de métodos de análisis visuales es una tarea muy desafiante debido a las problemáticas como: (1) Los sistemas de transporte público son cada vez más complejos para satisfacer el crecimiento de la población. (2) La mayoría de las técnicas existentes emplean métodos de visualización de red y estas técnicas se centran en mostrar la topología de la red en relación a las paradas, ignorando varios factores relacionados con la movilidad tales como: el tiempo de conducción, el tiempo de traslado, el tiempo de espera y los patrones de solo 24 horas. (3) En este artículo, los factores relacionados con la movilidad no son estáticos. Estos datos varían dinámicamente con el paso del tiempo y espacio.

El objetivo de los autores fue desarrollar una interfaz de análisis visual para presentar y explorar la movilidad de los pasajeros en un sistema de transporte público. Después de explorar diferentes diseños alternativos, los autores decidieron hacer una integración de datos, creando una solución de tres módulos de visualización: (1) Vista de un mapa isócrona para información geográfica. (2) Vista de un mapa de flujo isotime para la comparación y manipulación de información serie tiempo. (3) Vista de viajes pares (Origen-Destino) para el análisis visual detallada de los factores de movilidad a lo largo de las rutas entre pares de origen-destino.

En este trabajo utilizaron datos de las tarjetas personalizadas RFID de los pasajeros. Con ayuda de estas tarjetas los usuarios registran los viajes en el sistema de transporte público de Singapur durante un día laboral normal. Estos datos incluyen el sistema de metro llamado tránsito masivo rápido (MRT) y la red de autobuses públicos. El sistema lector de tarjetas registra cada acción de entrada y salida de usuarios, de igual manera este registro considera las transferencias entre el autobús y el servicio MRT.

En conclusión, los autores indican que para el desarrollo de su trabajo tuvo que explorar y comparar otros diseños de interfaces de análisis visual para la exploración de la movilidad, sin embargo, los diseños elegidos fueron los más apropiados para su trabajo de investigación. Además, propone la estrategia visual llamada rueda de movilidad, esto en la vista de viajes pares OD, con el fin de examinar la variación temporal de 24 horas del día con toda la información de movilidad involucrada.

3.2 Creación o el uso de conjuntos de datos de movilidad

4. Creación de un repositorio en FIWARE con datos de transporte público

En este trabajo de investigación [32] se establece que, en la actualidad, el flujo de la población de las áreas rurales a las áreas urbanas ha provocado un aumento descomunal de la población en las ciudades. Este crecimiento provoca que las personas que viven en grandes ciudades tengan estrés, una mala calidad de aire debido a la contaminación de los vehículos y una mala calidad de algunos servicios públicos.

Los problemas de las grandes ciudades no las puede resolver una sola persona, es por ello que los autores deciden aportar una solución de software que involucra los datos de movilidad de transporte público del Ayuntamiento de Valencia.

Un objetivo importante de este trabajo de investigación es la carga masiva de datos GTFS a la plataforma de datos FIWARE. Para cumplir este objetivo, los autores requirieron del uso de diversos entornos computacionales como el lenguaje de programación Phyton, gestor de base de datos MongoDB y Docker; con las cuales le permitieron crear repositorios en un servidor propio con conexión a los servicios de FIWARE, y mediante una serie de códigos y configuraciones computacionales, los autores lograron la carga de datos de forma masiva a la Plataforma FIWARE. Estos datos son obtenidos del portal de datos abiertos de la Empresa Municipal de Transporte (EMT) del Ayuntamiento de Valencia. Los datos obtenidos cuentan con el estándar GTFS, el cual maneja cinco ficheros de datos: (1) GtfsStops, indica las paradas que existen a lo largo de la ruta. (2) GtfsRoutes, indica las rutas de las distintas líneas. (3) GtfsAgency, define los viajes de cada ruta. (4) GtfsTrips: define los viajes de cada ruta. (5) GtfsStop_time, define los horarios de una parada determinada.

Puesto que los cinco ficheros GTFS son archivos de texto, los datos carecen de relaciones entre ellos. Por esta razón, los autores consideraron la creación de las relaciones de los ficheros mediante una base de datos relacional. Incluyendo estas relaciones, los autores señalan que es más eficiente enviar los datos a la Plataforma FIWARE. Como método de prueba, los autores utilizan ficheros GeJSON y con ayuda de la plataforma http://geojson.io/ consiguió ver la representación gráfica de cada parada de autobuses de la EMT de Valencia.

En conclusión, los autores indican que una vez cargados los datos en la Plataforma FIWARE, estos podrán ser utilizados por diferentes usuarios FIWARE y otros entornos para diversos fines. Uno de esos fines es la creación de una aplicación que permita ver, al usuario final, cuál de todas las paradas que tiene su alrededor o en un radio cercano es la más cercana. Además, en trabajos futuros se incluye la determinación de la ruta más cercana entre dos puntos o inclusive determinar la ruta más ecológica en cuanto al desplazamiento urbano.

5. Adquisición y análisis de datos del distanciamiento social durante la pandemia COVID-19

En este trabajo de investigación [33] se busca tener conocimiento relacionado con la concentración de la población urbana de los EE. UU., por motivo de la pandemia COVID-19. Uno de los hallazgos de este trabajo es que existe disparidad en la dinámica del distanciamiento según el ingreso económico de la persona. De acuerdo con los autores, los datos de movilidad humana brindan

información valiosa de la forma en cómo se ajustó el comportamiento de los viajes durante la pandemia.

Los autores recopilaron, procesaron y calcularon datos de movilidad de cuatro fuentes: (1) Informes de tendencia de movilidad de Apple. (2) Informes de movilidad de la comunidad de Google. (3) Datos de movilidad e Descartes Labs. (4) Movilidad de twitter calculada a través de la distancia ponderada.

El consumo de los datos para las dos primeras fuentes de información fue mediante archivos CSV con la información de la zona geográfica que se quiere analizar, por otro lado, el consumo de los datos de los sitios restantes fue mediante APIs de acceso a datos. La representación de la información obtenida de las diferentes fuentes es distinta para cada uno de los sitios (no cuentan con un estándar único), lo que lleva al autor a un desafío, el cual es estandarizar cada uno de los datos a fin de poder lograr el objetivo de hacer comparaciones entre cada uno de los sitios.

Los autores utilizan métodos estadísticos para la representación de los datos de movilidad, de esta forma fue posible visualizar los datos de las diferentes fuentes en una gráfica teniendo una mejor perspectiva para la comparación.

En conclusión, los autores describen que su estudio revela similitudes y disimilitudes de movilidad humana de las cuatro fuentes de datos, de igual manera su estudio indica la disparidad en la dinámica del distanciamiento acorde al ingreso económico de la persona o la zona geográfica donde habita.

A pesar de manejar diferentes formas de representación de los datos, el impacto de la pandemia COVID-19 está documentada de forma apropiada, sin embargo, el utilizar datos de diferentes fuentes con diferentes modelos de representación es un trabajo laborioso, ya que se aplicó una estandarización de datos.

6. Análisis y visualización de datos de diferentes fuentes para el estudio del distanciamiento durante la pandemia COVID-19

Tras el inicio de la pandemia de COVID-19 en el Reino Unido dio lugar a una serie de políticas de distanciamiento social. Este distanciamiento ha provocado una movilidad reducida en diferentes regiones. En este trabajo de investigación [34] se realiza un estudio relacionado con la movilidad urbana de algunas zonas del Reino Unido y verificar cómo fue cambiando la movilidad en el transcurso del tiempo.

Los datos a nivel multitudes sobre el uso de teléfonos móviles se pueden utilizar como un indicador de los patrones reales de movilidad de la población, además, estos datos proporcionan una forma de cuantificar el impacto de las medidas de distanciamiento social en los cambios de la movilidad. Los atores utilizan dos conjuntos de datos basados en teléfonos móviles, los cuales son: (1) Data for Good de Facebook proporciona datos sobre la cantidad diaria de usuarios activos en la aplicación con servicio de ubicación GPS, de igual forma proporciona datos relacionados con los viajes realizados por el usuario. (2) Datos anonimizados y agregados a nivel de multitudes de O2 comparte información de la ubicación en tiempo real con ayuda de la señal telefónica del celular.

Para permitir la comparación de los datos de la red móvil con los datos de Facebook los autores definieron un nivel de referencia de movilidad como el número medio diario de viajes dentro de una

semana determinada. Posteriormente, valida los datos sobre el número medio diario de viajes dentro de regiones espaciales. Este análisis de datos permitió evaluar los cambios de movilidad tanto en general como desglosado en áreas de alta y baja densidad poblacional.

Para lograr este estudio de datos, los autores desarrollaron un código en la herramienta R con el cual le permitió modelar y representar la información de forma gráfica, de tal forma que se pudo tener una visualización de los cambios de movilidad que paso durante la pandemia COVID-19. Así es como los autores demostraron que hubo reducción general de la movilidad, de igual forma, demostró que los dos conjuntos de datos diferentes producen tendencias similares, aunque con algunas diferencias especificas en la ubicación. Analizar los datos de diferentes fuentes con diferentes atributos fue una complicación técnica que encontraron los autores durante el desarrollo de este trabajo.

3.3 Conclusiones del estado del arte

En este capítulo se han descrito una serie de trabajos relacionados con el trabajo de investigación desarrollado en esta tesis. Los trabajos anteriormente descritos muestran diferentes métodos, herramientas y sistemas que se utilizan para el análisis de datos de la movilidad urbana. De igual manera describen las complejidades que se pueden presentar al momento de tratar datos de diferentes fuentes, ya que estos datos cuentan con diferentes atributos de representación. Sin embargo, se demostró que el análisis de datos de movilidad es un caso de estudio que brinda conocimiento valioso. Este conocimiento puede ayudar a la sociedad, economía, salud y a la ecología.

A partir de los trabajos de investigación descritos en el estado del arte, se generó una tabla comparativa con el propósito de tener un panorama general de las características de cada uno de los trabajos descritos y la investigación presentada en este trabajo de tesis. Los criterios tomados en cuenta son los siguientes:

- **Investigación:** indica el nombre de la investigación que se analizó.
- **Objetivo:** indica el objetivo de la investigación.
- **Datos utilizados:** indica las fuentes de datos utilizados y que tipo de datos son (históricos o en tiempo real)
- **Estandarización de datos:** indica sí se estandarizan los datos.
- **Datos que conservan:** indica si se conservan datos históricos o datos en tiempo real que permitan su reutilización
- **Representación gráfica de los datos:** indica sí los datos son representados de forma gráfica.

La tabla comparativa de los trabajos relacionados y nuestro trabajo de investigación se muestra en la Tabla 3.1.

Tabla 3.1 Comparativa de los trabajos relacionados y la tesis

Investigación	**Objetivo**	**Datos utilizados**	**Estandarización de datos**	**Datos que conservan**	**Representación gráfica de los datos**
Aplicación móvil SedVis para la prevención del sedentarismo [29]	Investigar el efecto de las visualizaciones del patrón de movilidad en la planificación de acciones de los usuarios para combatir el comportamiento sedentario.	Datos en tiempo real de APIs como: (1) Google Activity Recognition. (2) Google Maps. (3) Google Fit	No indica si se realiza una previa estandarización de datos	No conserva datos históricos ni en tiempo real	Si existe representación gráfica
Visualización de la movilidad urbana de la Ciudad de Sao Paulo [30]	Proponer una serie de visualizaciones de datos de movilidad urbana que ayuden al análisis y la toma de decisiones que mejore la movilidad en la zona estudiada.	Datos históricos de las encuestas Origen-Destino del metro metropolitano de Sao Paulo	No indica si se realiza una previa estandarización de datos	No conserva datos históricos ni en tiempo real	Si existe representación gráfica
Visualización de la movilidad del sistema de transporte público [31]	Desarrollar una interfaz de análisis visual para presentar y explorar la movilidad de los pasajeros en un sistema de transporte público.	Datos históricos de tarjetas personalizadas RFID del MRT de Singapur	No indica si se realiza una previa estandarización de datos	No conserva datos históricos ni en tiempo real	Si existe representación gráfica
Creación de un repositorio en FIWARE con datos de transporte público [32]	Crear un repositorio de datos de movilidad que contenga datos de transporte público urbano.	Datos históricos GTFS del transporte de Valencia	Si aplica estandarización de datos. Pero no genérica	Solo conserva datos históricos	Si existe representación gráfica
Adquisición y análisis de datos del distanciamiento social durante la pandemia COVID-19 [33]	Buscar conocimiento relacionado con la concentración de la población urbana de lo E. U., por motivos de la pandemia COVID-19.	Datos históricos de: (1) informes de tendencia de movilidad de Apple. (2) Informes de movilidad de Google. (3) Datos	Si aplica estandarización de datos. Pero no genérica	No conserva datos históricos ni en tiempo real	Si existe representación gráfica

		de Descartes Labs (4) Movilidad de Twitter			
Investigación	**Objetivo**	**Datos utilizados**	**Estandarización de datos**	**Datos que conservan**	**Representación gráfica de los datos**
Análisis de datos de diferentes fuentes para el estudio del distanciamiento durante la pandemia COVID-19 [34]	Realizar un estudio relacionado con la movilidad urbana de algunas zonas del Reino Unido y verificar como fue cambiando la movilidad en el transcurso del tiempo durante la pandemia COVID-19.	Datos en tiempo real de: (1) Data for Good de Facebook. (2) Datos anonimizados de O2 de la señal telefónica del lugar	No indica si se realiza una previa estandarización de datos	No conserva datos históricos ni en tiempo real	Si existe representación gráfica
Desarrollo de un espacio unificado de datos para movilidad en ciudades	Desarrollar un espacio unificado de datos que contenga datos en tiempo real e históricos de los Metrobuses y congestión vehicular de la CDMX.	Datos en tiempo real de: (1) Plataforma de Metrobuses CDMX (2) Congestión vehicular de Here Maps	Si aplica estandarización genérica de datos	Conserva datos en tiempo real y datos históricos	Si existe representación gráfica

Capítulo 4

Metodología de solución para el desarrollo del espacio unificado de datos

En este capítulo se describe la metodología de solución propuesta para la creación de un espacio unificado de datos de movilidad en ciudades. En la primera sección se describe de forma breve la metodología de solución y en las siguientes secciones se describen de manera detallada cada una de los procesos que la componen.

4.1 Descripción general de la metodología de solución

En este proyecto de tesis se desarrollaron unos Alimentadores de Datos de Movilidad Urbana (ADMU) que alimentan al espacio unificado con datos de la ubicación de los Metrobuses y congestión vehicular de la CDMX. Este espacio unificado junto con los Alimentadores ADMU se encuentran funcionando en un servidor local localizado en el SITE de CENIDET. El espacio unificado ayuda a la extracción y análisis de los datos de movilidad urbana de la Ciudad de México. Los procesos principales de la solución propuesta son presentados en la Figura 4.1.

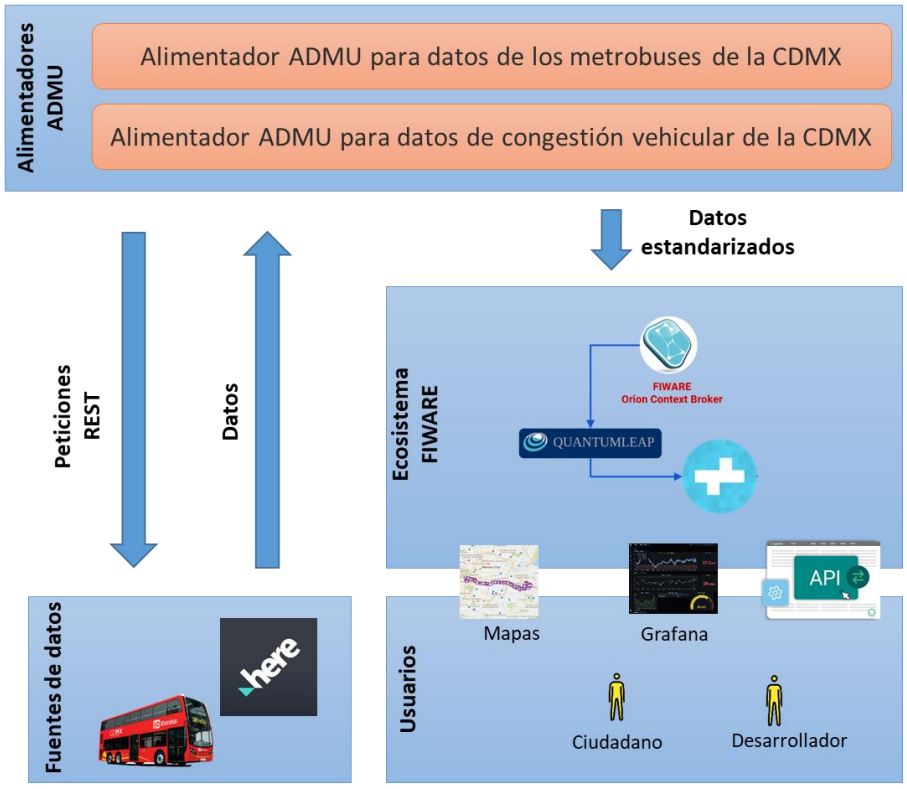


Figura 4.1 Vista general de la solución propuesta

El espacio unificado es capaz de conservar datos históricos y contener datos en tiempo real de la ubicación de los Metrobuses y la congestión vehicular de la Ciudad de México, de igual manera, contiene una herramienta para la monitorización de los datos, lo cual ayuda a identificar si existen problemas de monitoreo.

El espacio unificado mejora el procedimiento para la extracción de datos de movilidad de la Ciudad de México. Este espacio unificado utiliza las herramientas de la solución de FIWARE. Esta solución tecnológica está basada en software libre y tiene a su alcance las herramientas y estándares suficientes para el despliegue de una solución IoT relativamente sencilla, gratuita y modificable.

El principal y único componente obligatorio de la solución FIWARE es el *Orion Context Broker*, la cual permite almacenar únicamente los datos más actuales. De igual forma cuenta con el componente denominado Quantumleap el cual permite almacenar el histórico de los datos. Para que FIWARE pueda almacenar datos es necesario estandarizarlos en entidades NGSI. Este estándar entre otras cosas, brinda los atributos necesarios que se requieren para la representación de dispositivos IoT.

El espacio unificado es alimentado con los datos provistos por el Sistema de Corredores de Transporte Público de Pasajeros de la Ciudad de México, conocida como Metrobús, y por la plataforma de denominado HEREMaps el cual comparte información de la congestión vehicular de la Ciudad de México. Ambas plataformas comparten datos en tiempo real. Los datos extraídos por ambas plataformas son sometidos a reglas de mapeo para poder representar la entidad *Vehicle* y la entidad *TrafficFlowObserved*, respectivamente. Ambas entidades cuentan con el estándar NGSI.

Durante el desarrollo de este proyecto de tesis se crearon dos alimentadores ADMU que se encargan de extraer, mapear y enviar los datos al espacio unificado: (1) Alimentador que utiliza los datos de los Metrobuses. (2) Alimentador que utiliza los datos de congestión vehicular obtenidos de la Plataforma HEREMaps. Cada alimentador desarrollado tiene sus propias reglas de mapeo y sus métodos para la transformación de datos equivalentes.

Para la monitorización de los datos que se encuentran alojados en el espacio unificado propuesto se desarrollaron aplicaciones web las cuales consumen datos en tiempo real o datos históricos. Los datos consumidos son representados en un mapa donde se aprecia la ubicación de los Metrobuses y la congestión vehicular de la Ciudad de México. Asimismo, se desarrollan una serie de consultas tipo SQL dentro del componente Grafana para estudiar de forma gráfica el comportamiento de los datos de movilidad que se están guardando en nuestro espacio unificado.

El espacio unificado de datos de movilidad se desarrolló en cinco procesos (Figura 4.2).

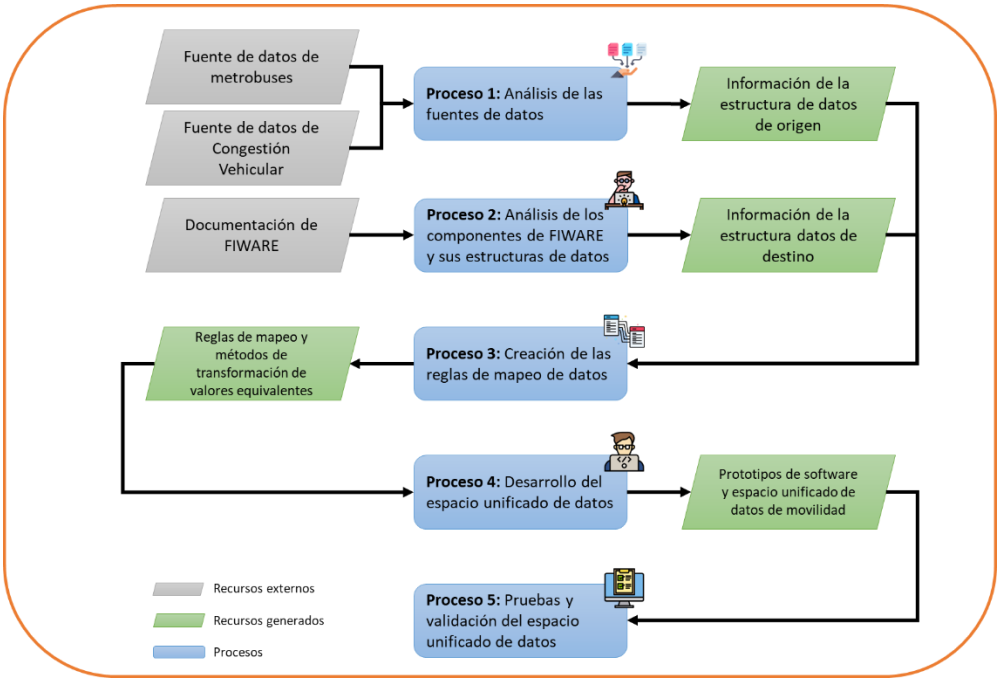


Figura 4.2 Metodología de solución para el desarrollo del espacio unificado de datos

Proceso 1: Análisis de las fuentes de datos

En este proceso se estudiaron las dos fuentes de datos: (1) Datos en tiempo real de la ubicación de los Metrobuses de la Ciudad de México. (2) Datos en tiempo real de la congestión vehicular de la Ciudad de México. De ambas fuentes de datos se estudió el procedimiento de la extracción de los datos, las estructuras de datos, los métodos para recorrer cada una de las instancias encontradas y el método para acceder a valores específicos de las instancias.

Proceso 2: Análisis de los componentes de FIWARE y sus estructuras de datos

En este proceso se llevó cabo un estudio sobre el funcionamiento de los componentes de la Plataforma FIWARE y los métodos para su implementación, asimismo se comprenden los estándares de los modelos de datos utilizados por FIWARE y se identifican los mejores modelos que se adapten a las necesidades de este proyecto de tesis.

Proceso 3: Creación de las reglas de mapeo de datos

En este proceso se crearon las reglas de mapeo para cada una de las estructuras de datos de origen: (1) Datos de los Metrobuses de la CDMX al modelo de datos *Vehicle*. (2) Datos de la congestión vehicular de la CDMX al modelo de datos *TrafficFlowObserved*. Al mismo tiempo se diseñaron los métodos para realizar transformaciones equivalentes de los valores de los atributos que lo requieran.

Proceso 4: Desarrollo del espacio unificado de datos

En este proceso se implementó un servidor local con las funcionalidades de los componentes de FIWARE. Al mismo tiempo se desarrollaron los alimentadores ADMU que se encargan de procesar los datos de los Metrobuses y congestión vehicular de la CDMX de forma independiente para poder ser alojados en el espacio unificado de datos.

Proceso 5: Pruebas y validación del espacio unificado de datos

En esta fase se desarrollaron una serie de aplicaciones web para la representar la ubicación de los Metrobuses o la congestión vehicular de la CDMX dentro de un mapa. Esta representación puede ser con datos en tiempo real o con datos históricos. De igual manera, se desarrollaron consultas tipo SQL para la representación gráfica dentro del componente Grafana.

A continuación, se presenta el desarrollo detallado de cada uno de estos 5 procesos claves del proyecto.

4.2 Proceso 1: Análisis de las fuentes de datos

El primer proceso de nuestra metodología de solución consiste en el análisis de las dos fuentes de datos utilizadas en este proyecto de tesis: (1) Fuente de datos de los Metrobuses de la Ciudad de México. (2) Fuente de datos HEREMaps, la cual comparte datos de la congestión vehicular de la Ciudad de México. La Figura 4.3 se muestra los puntos que son analizados de cada una de las fuentes de datos. Cabe mencionar que los procesos para analizar los datos tienen el mismo nombre, sin embargo, durante su implementación es diferente para cada una de las fuentes. La sección 4.2.1

presenta el análisis de la fuente de datos de los Metrobuses y la sección 4.2.2 presenta el análisis de la fuente de datos de congestión vehicular.

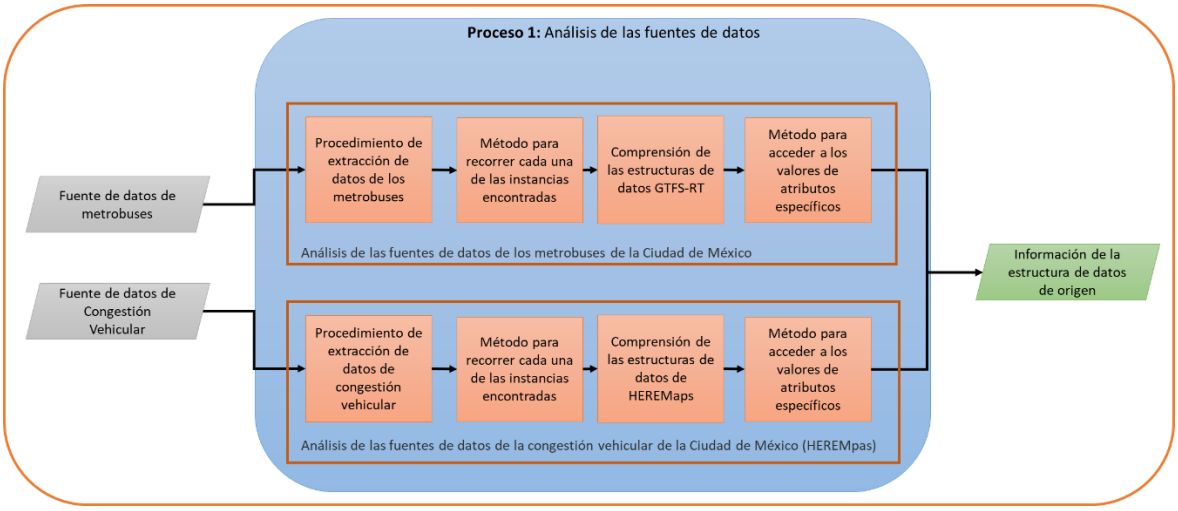


Figura 4.3 Proceso del análisis de las fuentes de datos

4.2.1 Análisis de la fuente de datos de los Metrobuses de la Ciudad de México

Procedimiento de extracción de datos de los Metrobuses de la Ciudad de México

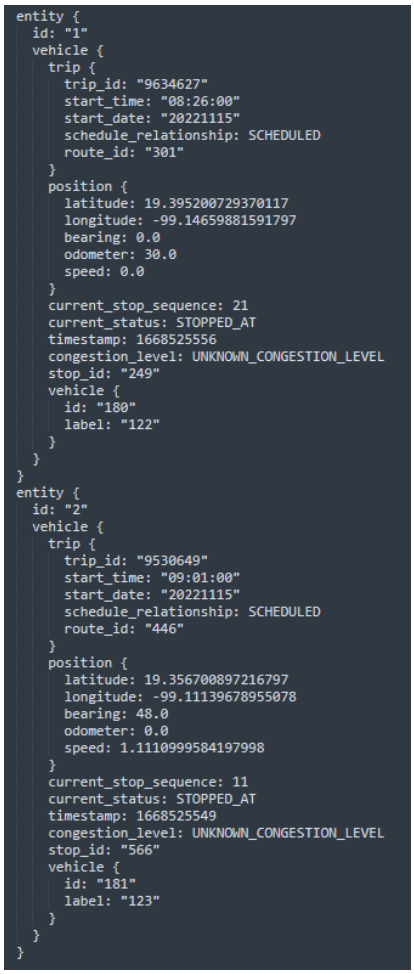
Para poder extraer datos en tiempo real de los Metrobuses de la Ciudad de México se realiza la solicitud correspondiente a los administradores del Sistema de Corredores de Transporte Público de Pasajeros de la Ciudad de México. Estos administradores comparten las ligas de acceso a los datos, las claves de acceso y la documentación para poder trabajar con los datos en formato GTFS-RT.

La biblioteca de google.transit es necesaria para trabajar en el procesamiento de los datos en formato GTFS-RT. Esta biblioteca ayuda al procesamiento de los datos GTFS-RT. De acuerdo a la documentación compartida por los propietarios del sitio se deben instalar los paquetes de acuerdo al lenguaje de programación que se esté utilizando. La agencia propietaria de los datos especificó el lenguaje de programación Python como lenguaje de referencia de los datos en formato GTFS-RT.

El formato datos GTFS-RT brinda tres diferentes tipos de información en tiempo real: (1) Datos de las alertas del servicio. (2) Datos sobre la actualización de los viajes. (3) Datos de la ubicación de las unidades. En nuestro proyecto de tesis se ocupan únicamente los datos de la ubicación en tiempo real de las unidades.

Teniendo en cuenta que se tiene instalada la librería para procesar datos GTFS-RT, se identifica la liga de acceso a los datos en tiempo real de la ubicación de los Metrobuses y se tienen las claves de acceso enseguida se desarrolla una función para la extracción de los datos de la ubicación de los Metrobuses, esta función se denominada “consumeGTFS”. En la ejecución de esta función se obtiene una lista que contiene los datos de la ubicación de todos los Metrobuses que se encuentran reportando sus datos al momento de la consulta, aproximadamente 700 Metrobuses.

Esta lista es almacenada en un objeto denominado “feed” para su tratamiento. En la Figura 4.4 se observa un segmento con dos instancias de la lista que se obtiene de este proceso.



```
entity {
  id: "1"
  vehicle {
    trip {
      trip_id: "9634627"
      start_time: "08:26:00"
      start_date: "20221115"
      schedule_relationship: SCHEDULED
      route_id: "301"
    }
    position {
      latitude: 19.395200729370117
      longitude: -99.14659881591797
      bearing: 0.0
      odometer: 30.0
      speed: 0.0
    }
    current_stop_sequence: 21
    current_status: STOPPED_AT
    timestamp: 1668525556
    congestion_level: UNKNOWN_CONGESTION_LEVEL
    stop_id: "249"
    vehicle {
      id: "180"
      label: "122"
    }
  }
}
entity {
  id: "2"
  vehicle {
    trip {
      trip_id: "9530649"
      start_time: "09:01:00"
      start_date: "20221115"
      schedule_relationship: SCHEDULED
      route_id: "446"
    }
    position {
      latitude: 19.356700897216797
      longitude: -99.11139678955078
      bearing: 48.0
      odometer: 0.0
      speed: 1.1110999584197998
    }
    current_stop_sequence: 11
    current_status: STOPPED_AT
    timestamp: 1668525549
    congestion_level: UNKNOWN_CONGESTION_LEVEL
    stop_id: "566"
    vehicle {
      id: "181"
      label: "123"
    }
  }
}
```


Figura 4.4 Segmento de la lista con los datos de dos Metrobuses de la CDMX

Método para recorrer la lista para identificar cada una de las instancias encontradas

La ejecución de la función “consumeGTFS” extrae una lista con los datos en tiempo real de la ubicación de todos los Metrobuses de la Ciudad de México. Esta lista es almacenada en el objeto denominado “feed”. Puesto que son los datos de todos los Metrobuses en una sola lista, se implementa un método para identificar un Metrobús a la vez. De esta manera poder extraer la información que se requiere por unidad.

Con el objetivo de identificar cada unidad de la lista de Metrobuses obtenida se utiliza el siguiente código.:

1) Metrobus = feed.entity[i]

En donde el valor “i” indica la posición de la instancia que tiene dentro de la lista, este valor comienza a partir del número cero. Los datos obtenidos de un Metrobús de forma individual se guardan en el objeto denominado “Metrobús”. En la Figura 4.5 se observa la información correspondiente a un solo Metrobús.

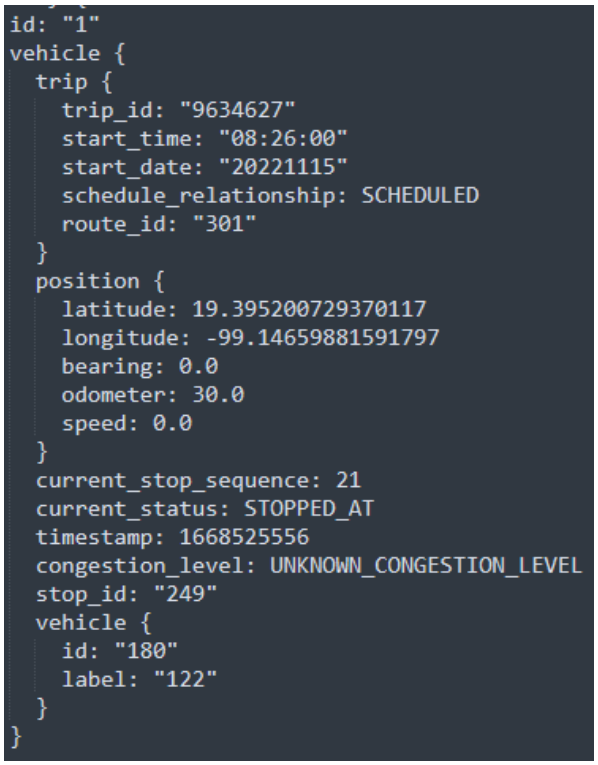


Figura 4.5 Ejemplo de los datos de un solo Metrobús

Comprensión de la estructura de datos de los Metrobuses de la Ciudad de México

El objeto denominado "Metrobus" contiene los datos de un Metrobús en específico. En la Figura 4.5 se aprecian la estructura de datos que utiliza el formato GTFS para la representación de una unidad de transporte público. En esta estructura se pueden observar cuatro apartados principales, tal y como se puede apreciar en la Figura 4.6:

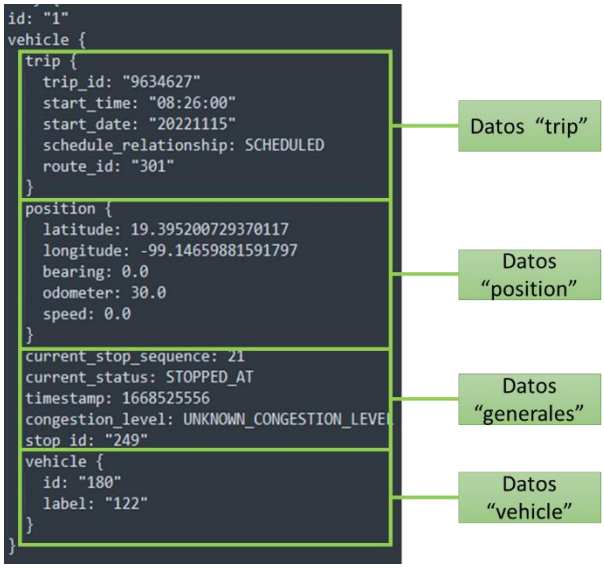


Figura 4.6 Apartados principales de la estructura de datos que utiliza GTFS

Datos generales

Este apartado brinda datos generales del Metrobús. Entre los datos compartidos en este apartado se encuentran.

a) current_stop_sequence: brinda el número de secuencia de la parada.
b) current_status: brinda un valor numérico que indica el estatus del vehículo. Pueden ser el valor 1 “STOPPED_AT” o el valor 2 “IN_TRANSIT_TO”.
c) timestamp: brinda la hora y fecha en el que se extrae la información.
d) congestion_level: brinda el nivel de congestión vehicular. Actualmente desactivado para los datos GTFS-RT de los Metrobuses de la Ciudad de México.
e) route_id: brinda el id de la ruta que está siguiendo la unidad.

Datos “trip”

Este apartado brinda información general del viaje que está realizando el Metrobús. En nuestro proyecto de tesis se decide no utilizar este apartado.

Datos “position”

Este apartado brinda datos sobre la ubicación en tiempo real del Metrobús que se está monitoreando. Los datos compartidos son los siguientes.

a) latitude: brinda un valor decimal que representa la latitud del Metrobús.
b) longitude: brinda un valor decimal que representa la longitud del Metrobús.
c) bearing: brinda un valor decimal entre 0 y 360 donde 90 es norte. Este valor representa la dirección hacia donde se está dirigiendo el Metrobús.
d) odometer: brindas un valor decimal que representa el odómetro del Metrobús. Este valor se encuentra en metros.
e) speed: brinda un valor decimal que representa la velocidad que tiene el Metrobús. Este valor se encuentra en metros sobre segundo (m/s).

Datos Vehicle

Este apartado brinda información sobre la unidad.

a) id: brinda un valor entero que representa el id del Metrobús en el contexto del sistema computacional.
b) label: brinda un valor entero que representa el número económico del Metrobús.

Método para acceder a los valores de atributos específicos

En este proyecto de tesis no se utilizan todos los datos obtenidos por la función “consumeGTFS”. Por esta razón se identifican los métodos necesarios para el aislamiento de los valores de atributos específicos. Para este procedimiento se utiliza el objeto denominado “Metrobus”, el cual contiene únicamente los datos de un solo Metrobús.

En la Figura 4.7 se observan cuatro ejemplos de la extracción del valor de cuatro atributos distintos: (1) Extracción del valor de longitud. (2) Extracción del valor de velocidad. (3) Extracción del valor de estatus. (4) Extracción del número económico de la unidad.

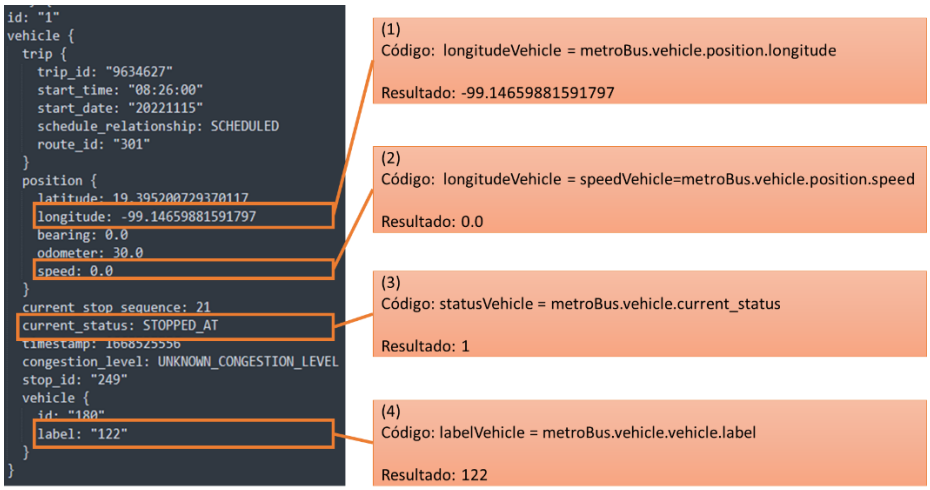


Figura 4.7 Ejemplos de extracción de valores específicos

4.2.2 Análisis de la fuente de datos de la congestión vehicular de la Ciudad de México

Procedimiento de extracción de datos de la congestión vehicular de la Ciudad de México

La Plataforma HEREMaps genera de forma única contenido relacionado a la movilidad en ciudades y software que permite a las personas, las empresas y los gobiernos tener acceso a información de este tipo. Uno de los servicios que ofrece esta plataforma es brindar APIs de acceso rápido a datos sobre congestión vehicular.

Para poder extraer datos en tiempo real de la congestión vehicular se realiza el registro correspondiente a la Plataforma HEREMaps. Este registro nos brinda acceso a las diversas herramientas que tiene la plataforma. Durante la exploración de la plataforma se identifica que cuentan con múltiples APIs de acceso rápido a datos. Para nuestro proyecto de tesis se utiliza únicamente el API que comparte información de la congestión vehicular de zonas geográficas específicas.

Es importante comentar que este procedimiento para la extracción de datos para una ciudad en particular no se encuentra documentado en ningún manual, por lo cual forma parte de una de las contribuciones de esta tesis. La obtención de datos no es trivial y por tanto es detallada en esta sección.

La liga del API seleccionada para las necesidades de nuestro proyecto de tesis es la siguiente.

https://traffic.ls.hereapi.com/traffic/6.2/flow.json?**bbox=BBOX**&**responseattributes=sh**&**apiKey=APIKEY**

El API tiene tres variables principales que permiten la extracción de datos de congestión vehicular únicamente de la Ciudad de México: bbox, APIKey y responseattributes. Estas variables se describen a continuación.

bbox

Es un conjunto de dos coordenadas geográficas. Estas coordenadas deben de cubrir la mayor área posible de la zona que se quiere analizar. En este proyecto de tesis se enfoca en la zona de la Ciudad de México. El API requiere un orden especifico de las coordenadas. Este orden es: *latitud$_1$ y longitud$_1$; latitud$_2$ y longitud$_2$*.

El primer par de coordenadas pertenece a la esquina superior izquierdas de la CDMX, mientras que, el segundo par de coordenadas pertenecen a la esquina inferior derecha de la CDMX, las dos esquinas restantes se autocompletan. En la Figura 4.8 se representa un bbox cubriendo a la Ciudad de México. Las dos coordenadas utilizadas para cubrir a la Ciudad de México son 19.725181,-99.316750;19.061956,-98.788376

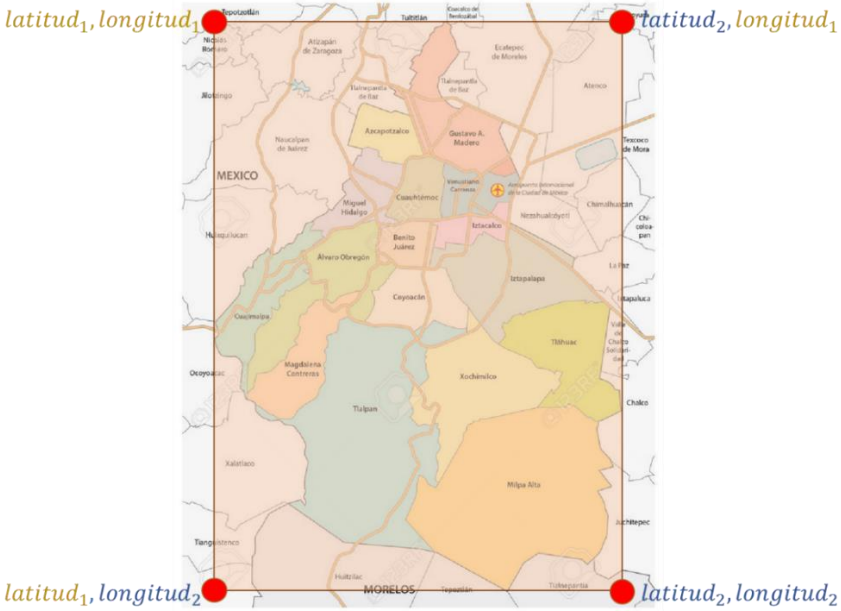


Figura 4.8 Ejemplificación del bbox

APIKey

Es una clave se seguridad que permite el acceso a los datos de congestión vehicular que se encuentran alojados en las APIs de la Plataforma HEREMaps.

responseattributes

Es un parámetro el cual solicita información adicional a la que comparte el API por sí sola. Para nuestro proyecto de tesis se utiliza el parámetro de entrada “sh”. Este parámetro devuelve las coordenadas del segmento de la calle, avenida o boulevard de la Ciudad de México.

Contemplando los valores de los parámetros bbox, APIkey y responseattributes se genera la liga que permite la adquisición de datos de congestión vehicular de la Ciudad de México. Esta liga se presenta a continuación.

https://traffic.ls.hereapi.com/traffic/6.2/flow.json?**bbox=19.725181,-99.316750;19.061956,-98.788376**&**apiKey=(MiApiKey)** &**responseattributes=sh**

Una vez que la liga se encuentra estructurada con los parámetros adecuados se desarrolla la función que extrae datos de congestión vehicular. Esta función se denomina "funcionConsumirDatosHereMaps". En la ejecución de esta función se obtiene una lista que contiene datos de congestión vehicular de los segmentos de la Ciudad de México. Se detectan un total de 4,133 segmentos que corresponden a calles y avenida de la CDMX.

Cabe mencionar que no existen una correspondencia 1 a 1 entre un segmento y una calle, por ejemplo, una calle larga, como es el caso de la Avenida Insurgentes se divide en múltiples segmentos. Existen calles pequeñas que si pueden ser especificadas con un solo segmento.

Estos datos son guardados en una lista JSON denominada "dataHereJson" para su tratamiento. En la Figura 4.9 se observan dos segmentos de la lista obtenida por la función desarrollada.


```
{
    "TMC": {
        "PC": 4121,
        "DE": "Viaducto Rio de la Piedad",
        "QD": "+",
        "LE": 0.01535
    },
    "SHP": [
        {
            "value": [
                "19.40684,-99.07859 19.40674,-99.07851 "
            ]
        },
        {
            "value": [
                "19.40674,-99.07851 19.40665,-99.07847 "
            ]
        }
    ],
    "CF": [
        {
            "TY": "TR",
            "SP": 11.74,
            "SU": 11.74,
            "FF": 12.68,
            "JF": 0.0,
            "CN": 0.71
        }
    ]
},
```

```
"TMC": {
    "PC": 4119,
    "DE": "Av Canal de Tezontle",
    "QD": "+",
    "LE": 0.65704
},
"SHP": [
    {
        "value": [
            "19.3921,-99.0878 19.39065,-99.08876 "
        ]
    },
    {
        "value": [
            "19.39065,-99.08876 19.39056,-99.08882 19.38734,-99.09088 19.38582,-99.09186 "
        ]
    },
    {
        "value": [
            "19.38582,-99.09186 19.38553,-99.09205 "
        ]
    },
    {
        "value": [
            "19.38553,-99.09205 19.38435,-99.09279 "
        ]
    },
    {
        "value": [
            "19.38435,-99.09279 19.38407,-99.09297 "
        ]
    },
    {
        "value": [
            "19.38407,-99.09297 19.38395,-99.09305 "
        ]
    }
],
"CF": [
    {
        "TY": "TR",
        "SP": 22.99,
        "SU": 22.99,
        "FF": 26.97,
        "JF": 1.3108,
        "CN": 0.7
```


Figura 4.9 Segmento de la lista con los datos de dos segmentos de la CDMX

Método para recorrer cada una de las instancias encontradas

Con la ejecución de la función "funcionConsumirDatosHereMaps" se extrae una lista con estructura JSON con los datos en tiempo real de la congestión vehicular de la Ciudad de México. Esta lista es almacenada en la variable denominada "dataHereJson". Puesto que son los datos de congestión vehicular de todos los segmentos de la CDMX, se implementa un método para identificar un segmento de la ciudad a la vez, el cual permita extraer la información que se requiere por cada uno de los segmentos de la Ciudad de México.

Para poder identificar un solo segmento de la ciudad se ejecuta la siguiente liga de código.

2) datos = dataHereJson['RWS'][i]['RW'][j]['FIS'][k]['FI'][l]

En donde los valores “i”, “j”, “k”, “l” son valores numéricos que indican la posición donde se encuentra cada uno de los segmentos de la ciudad. Estos valores comienzan a partir del número cero. Por otro lado, los valores “RWS”, “RW”, “FIS”, “FI” son los nombres de los indicadores de cada uno de los apartados que tiene la lista. Los datos obtenidos de un segmento de la Ciudad de México se guardan en la variable denominada “datos”. En la Figura 4.10 se observa la información correspondiente a la congestión vehicular de un solo segmento de la CDMX.


```
{
    "TMC": {
        "PC": 4121,
        "DE": "Viaducto Rio de la Piedad",
        "QD": "+",
        "LE": 0.01535
    },
    "SHP": [
        {
            "value": [
                "19.40684,-99.07859 19.40674,-99.07851 "
            ]
        },
        {
            "value": [
                "19.40674,-99.07851 19.40665,-99.07847 "
            ]
        }
    ],
    "CF": [
        {
            "TY": "TR",
            "SP": 11.74,
            "SU": 11.74,
            "FF": 12.68,
            "JF": 0.0,
            "CN": 0.71
        }
    ]
},
```


Figura 4.10 Ejemplo de los datos congestión vehicular de un solo segmento de la CDMX

Comprensión de la estructura de datos de la congestión vehicular de la Ciudad de México

En la variable denominada “datos” contiene los datos de la congestión vehicular de un solo segmentos de la Ciudad de México. En la Figura 4.10 se aprecia la estructura de datos que utiliza la Plataforma HEREMaps para la representación de la congestión vehicular de un segmento de la ciudad especifico. En esta estructura se pueden observar tres apartados principales, tal y como se puede apreciar en la Figura 4.11.

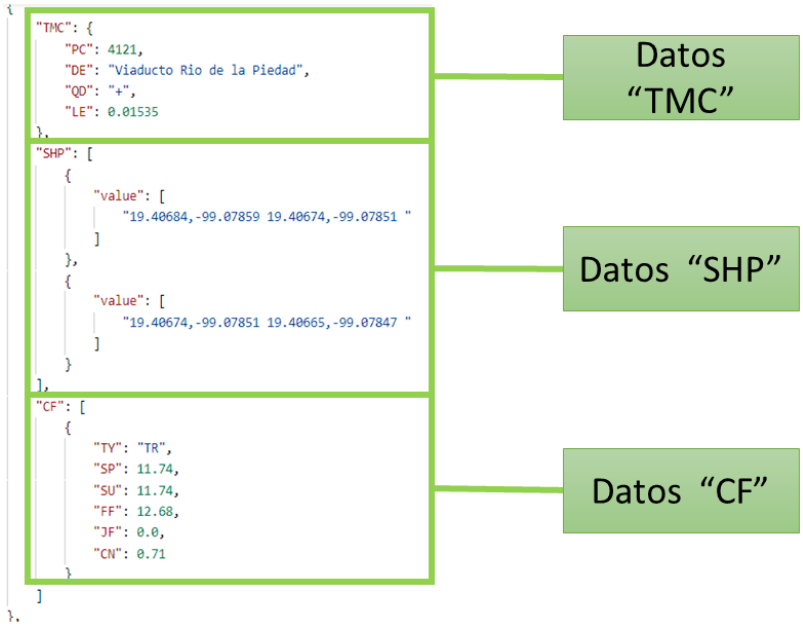


Figura 4.11 Apartados principales de la estructura de los datos de congestión vehicular

Datos "TMC"

Este apartado brinda datos generales del segmento de la ciudad. Entre los datos compartidos en este apartado se encuentra.

a) PC: brinda el id que le corresponde al segmento.
b) DE: brinda el nombre del segmento que se está analizando.
c) QD: brinda el sentido del tráfico que tiene el segmento de la ciudad. Los valores que pueden ser "-" o "+".
d) LE: brinda la longitud del segmento.

Datos "SHP"

Este apartado brinda las coordenadas del segmento de la calle, avenida o boulevard. En este apartado se aprecia que tiene múltiples "values" los cuales cada uno representa un vector dentro del segmento de la ciudad.

En la Figura 4.12 se puede observar las coordenadas extraídas del apartado SHP representadas en un mapa. En este mapa se pueden apreciar dos vectores de color azul. Estos vectores representan el largo del segmento que se está analizando.

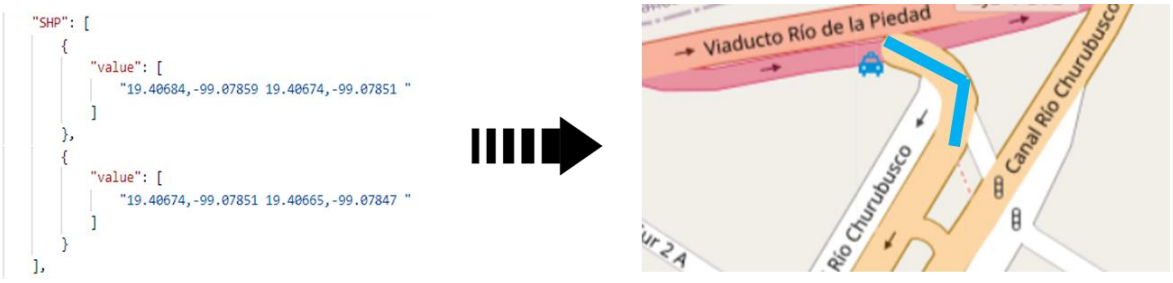


Figura 4.12 Ejemplo 1 de la representación de los datos SHP en un mapa

Es importante comentar que el número de vectores que comparte el apartado SHP es distinto para cada uno de los segmentos. Esta distinción depende del largo del segmento de la ciudad que se está analizando. Un ejemplo de esta distinción se presenta en la Figura 4.13 donde se puede apreciar que el número de "values" es mayor que el número de "values" de la Figura 4.12. Estos "values" se representan en un mapa y se nota que el segmento es considerablemente más largo. De igual forma, en este mapa se pueden apreciar ocho vectores de color azul. Estos vectores representan el largo del segmento que se está analizando.

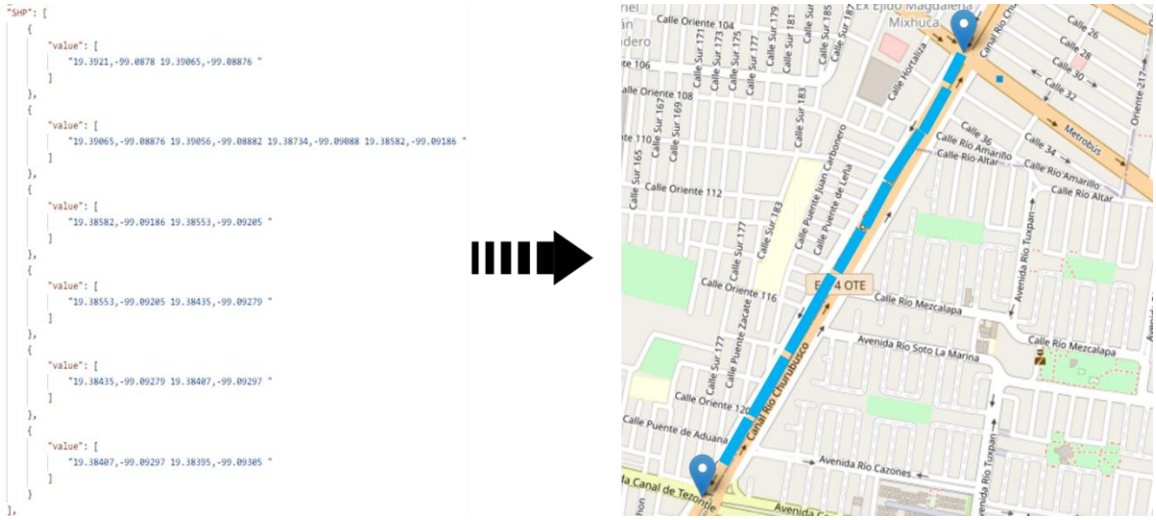


Figura 4.13 Ejemplo 2 de la representación de los datos SHP en un mapa

Datos "CF"

Este apartado brinda información sobre el flujo de la congestión vehicular del segmento que se está presentando en tiempo real. Los elementos del segmento son los siguientes:

a) TY: se utiliza cuando es necesario diferencia entre diferentes tipos de ubicación.
b) SP: brinda la velocidad promedio del flujo de tráfico que tiene el segmento de la ciudad limitado por el límite de velocidad. Este valor se encuentra en la escala Km/h.
c) SU: brinda la velocidad promedio del flujo de tráfico que tiene el segmento de la ciudad NO limitado por el límite de velocidad. Este valor se encuentra en la escala Km/h.
d) FF: brinda la velocidad de flujo libre en este segmento de la carretera.
e) JF: brinda el factor de atasco. Los valores de esta variable son numéricos del 0 al 10, en donde a medida que el valor se acerca al número 10, el factor de atasco empeora. Mientras

que, si el valor es 10 indica que la calle se encuentra cerrada. Si el valor obtenido es -1 indica que no se puede obtener la información.

f) CN: brinda un valor número que representa la confiabilidad de los datos. El valor de esta variable se encuentra entre el 0 y 1. Si el valor se encuentra cerca del 1, indica que los datos son en tiempo real, mientras que, si el valor se acerca al cero, indica que utiliza datos históricos. Si el valor obtenido es -1 indica que la lectura está cerrada.

Método para acceder a los valores de atributos específico

En este proyecto de tesis no se utilizan todos los datos obtenidos por la función "funcionConsumirDatosHereMaps". Por esta razón se identifican los métodos necesarios para el aislamiento de los valores de atributos específicos. Para este procedimiento se utiliza la lista denominada "datos" que contiene únicamente los datos de un solo segmento de la Ciudad de México.

En la Figura 4.14 se observan cuatro ejemplos de la extracción del valor de cuatro atributos distintivo: (1) Extracción del nombre del segmento. (2) Extracción de las coordenadas del segmento. (3) Extracción de la velocidad promedio del tráfico en el segmento. (4) Extracción del valor de la calidad de los datos obtenidos.

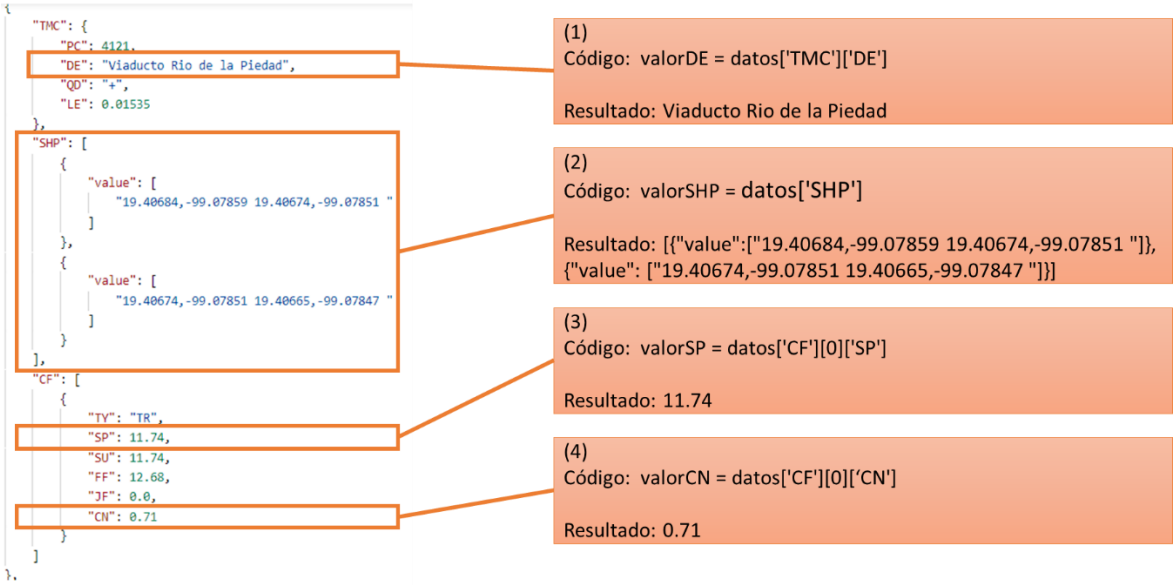


Figura 4.14 Ejemplos de extracción de valores específicos

4.3 Proceso 2: Análisis de los componentes de FIWARE y sus estructuras de datos

En este proceso se realiza un reconocimiento de los diferentes componentes de la Plataforma FIWARE, asimismo se investigan las herramientas y procedimientos necesarios para la implementación de un servidor con las funcionalidades de los componentes de FIWARE. Por otro lado, se comprende el estándar de los modelos de datos utilizados por la Plataforma FIWARE y se identifican los dos mejores modelos que se adaptan para la representación de un vehículo y la

congestión vehicular de un segmento de la Ciudad de México. En la Figura 4.15 se observa las actividades desarrolladas en este proceso.

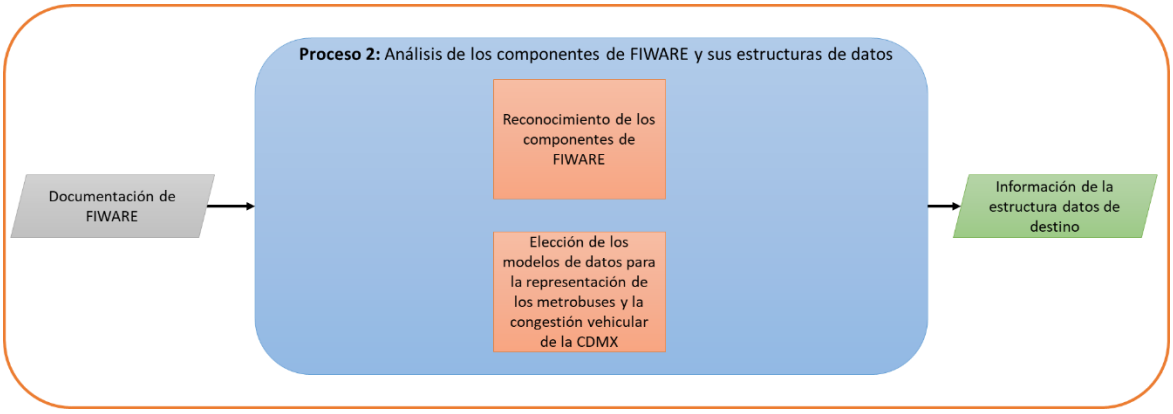


Figura 4.15 Proceso para el análisis de los componentes de FIWARE y sus estructuras de datos

4.3.1 Reconocimiento de los componentes de FIWARE

La herramienta FIWARE brinda una serie de soluciones enfocadas a la administración de datos de contexto que comparten los diversos dispositivos que conforman el Internet de la Cosas. En nuestro proyecto de tesis se utiliza la solución denominada Quantumleap [21].

Componente Quantumleap

Quantumleap es un servicio REST que permite almacenar, consultar y recuperar datos espacio-temporales. Quantumleap convierte los datos semiestructurados a un formato tabular y los almacena en una base de datos de series temporales. Los clientes que requieren extraer datos de esta solución pueden aplicar filtros de tiempo o de entidades. Estos filtros son aplicados de acuerdo a los datos que requiera el cliente.

Este componente tiene la funcionalidad de conservar los datos históricos de cada una de las entidades que se estén monitoreando.

El mecanismo para el almacenamiento de datos históricos dentro de Quantumleap es de forma transparente. Esta transparencia se logra gracias a la liga guardada en la variable “http” que se encuentra ubicado en la suscripción. Esta liga comunica el OCB con el Quantumleap por el puerto 8868.

El funcionamiento del mecanismo para el almacenamiento de los datos históricos dentro de Quantumleap consiste en que cada vez que el OCB reciba una actualización de datos, los datos que tiene el OCB durante de actualización los envía a Quantumleap para que el OCB contenga únicamente el dato más actual.

Cabe mencionar que el componente obligatorio para cualquier solución implementada por FIWARE es el *Orion Context Broker*. En la Figura 4.16 Arquitectura de la solución Quantumleap de FIWARE se muestra la arquitectura y los componentes de la solución Quantumleap [35]. Gracias a estos componentes hacen que la solución Quantumleap sea la apropiada para nuestro proyecto de tesis,

ya que esta solución tiene las facilidades para el almacenamiento de datos en tiempo real y datos históricos.

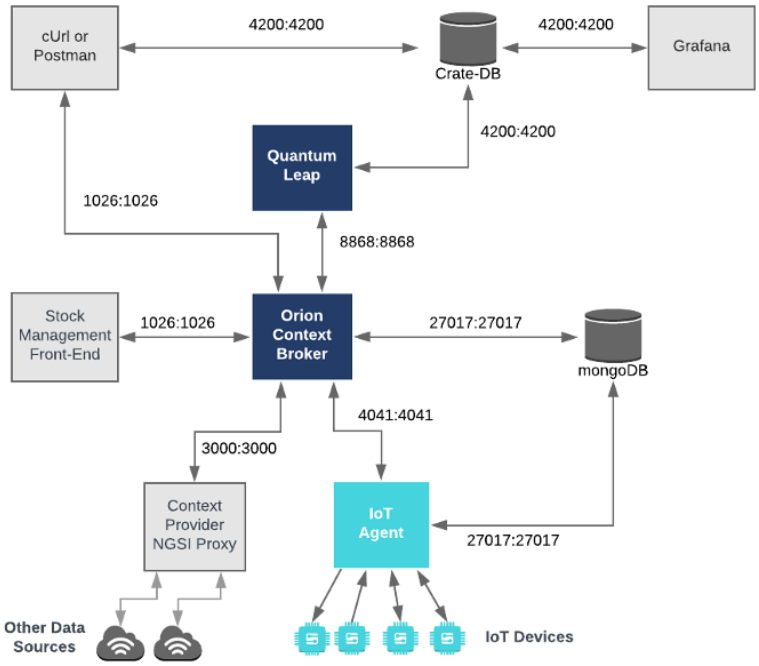


Figura 4.16 Arquitectura de la solución Quantumleap de FIWARE

La solución de Quantumleap contiene distintos componentes y cada uno de ellos tiene su respectiva funcionalidad. En este proyecto de tesis se utilizan únicamente los siguientes componentes.

Componente Orion Context Broker y mongoDB

El componente denominado *Orion Context Broker* (OCB) es obligatorio para cualquier solución propuesta por la Plataforma FIWARE. Este componente aporta la función fundamental en cualquier solución inteligente: administrar la información de contexto, consultarla y actualizarla.

El OCB permite la publicación de información de contexto por entidades. Una entidad es la representación lógica de un objeto y/o sensor que comparte información de sí mismo o de su entorno. Esta representación está basada en el estándar NGSI. Este estándar brinda las propiedades necesarias para la representación de las distintas entidades que se pueden encontrar en mundo del IoT. Los datos estandarizados facilitan su análisis y comprensión.

El OCB junto con el gestor de bases de datos mongoDB permiten almacenar únicamente los datos más actuales compartidos por las entidades y de igual forma almacena una suscripción que permite la actualización de los datos de cada entidad. Se tiene que contar con una suscripción por cada tipo de entidad que se quiere monitorear. En la Figura 4.17 se observa la estructura de la suscripción de entidades.

```
{
    "description":"",
    "subject":{
        "entities":[{
            "idPattern":"",
            "type":""
        }
        ],
        "condition":{
            "attrs":[
                ""
            ]
        }
    },
    "notification":{
        "attrs":[
            "",
            "",
            "",
            ""
        ],
        "http":{
            "url":"http://quantumleap:8668/v2/notify"
        },
        "metadata":[
            "",
            ""
        ]
    }
}
```


Figura 4.17 Estructura de la suscripción de entidades

La estructura de la suscripción almacena la siguiente información.

a) description: se ingresa una pequeña descripción sobre la entidad
b) idPattern: se ingresa el formato del id de la entidad. Ejemplo, id:sensor:*. En donde el carácter "*" puede cambiar por un numero consecutivo o un valor que pueda diferenciar las entidades. Ejemplo, id:sensor:2, id:sensor:34.
c) type: se ingresa el tipo de entidad la cual va a sufrir las actualizaciones.
d) condition.attrs: se ingresa el nombre del atributo de la entidad que debe de sufrir un cambio para que pueda actualizar todos los datos de la misma entidad. Este se conoce también como la alerta.
e) notification.attrs: se ingresan los nombres de los atributos que van a sufrir la actualización.
f) http: se ingresa la liga donde guardaran los datos históricos de las entidades.
g) metadata: se ingresan los atributos que fungirán como metadatos. Se considera ocupar la fecha de creación y actualización como metadatos.

Cabe mencionar que para la creación de la suscripción, creación y actualización de entidades se utilizan peticiones REST por el puerto 1026. Esencialmente las peticiones más utilizadas por este proyecto de tesis son POST para crear, PACH para actualizar y GET para consultar.

Componente CrateDB

Este componente es una base de datos SQL distribuida que simplifica el almacenamiento y el análisis de cantidades masivas de datos. CrateDB ofrece los beneficios de una base de datos SQL, la escalabilidad y flexibilidad típicamente asociada con las bases de datos NoSQL.

Dentro de la Plataforma FIWARE funge como un traductor de datos. El proceso de traducción consiste en convertir los datos semiestructurados de NGSI en forma tabular y los almacena en una base de datos de series tiempo, asociando cada registro de la base de datos con un índice de tiempo y, si está presente en los datos NGSI, una ubicación geográfica en la Tierra.

La comunicación entre Quantumleap y CrateDB es mediante el puerto 4200.

Componente Grafana

Este componente es una herramienta de visualización que se puede utilizar para mostrar gráficas de los datos de las entidades alojados en Quantumleap.

Para que Grafana pueda tener acceso a los datos alojados en Quantumleap, debe de existir una conexión por el puerto 5432 hacia el componente CrateDB. En la Figura 4.18 se observa el ejemplo de la configuración de la conexión entre Grafana y CrateDB. Cabe mencionar que inicialmente el tipo de base de datos que se configura dentro de Grafana es Postgres.

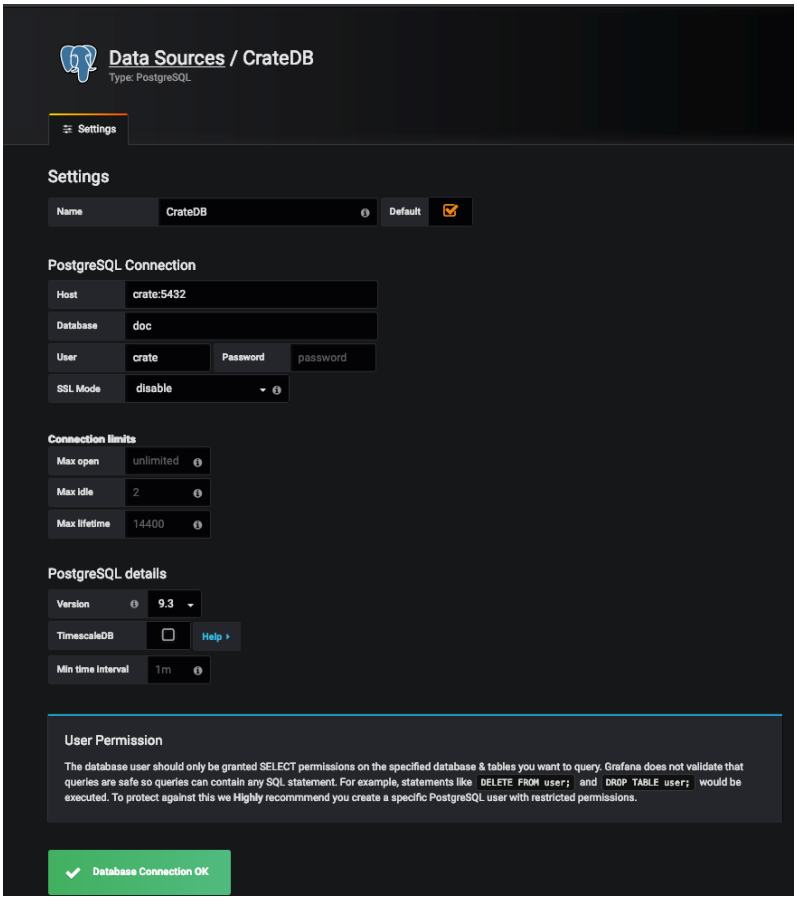


Figura 4.18 Ejemplo conexión entre Grafana y CrateDB

Lo valores que se deben de agregar son:

a) Nombre: se agrega el nombre de la conexión que se va a realizar. Puede ser cualquier nombre.
b) Host: se agrega la dirección URL donde se implementó CrateDB. En un ambiente de desarrollo local se agrega “crate:5432”.
c) Base de datos: se agrega el nombre del esquema de la base de datos.
d) Usuario: se agrega el usuario de acceso a la base de datos. De forma predeterminada es “crate”.
e) Modo SSL: se desactiva.

Con esta configuración se logra la conexión entre los componentes Grafana y CrateDB. Para generar visualizaciones gráficas dentro de Grafana se deben de crear *dashboards* y dentro de los *dashboards* se agregan las gráficas. Cada gráfica es generada con ayuda de una consulta de tipo SQL. Esta consulta se crea a partir de lo que se quiera observar o monitorear.

Para la implementación del servidor con las funcionalidades de cada uno de los componentes de un servidor FIWARE se puede observar el anexo A denominado Implementación de un servidor FIWARE.

4.3.2 Elección de los modelos de datos para la representación de los Metrobuses y congestión vehicular de la CDMX

Los modelos de datos estandarizados facilitan la compresión del objeto que se trata de representar, asimismo, beneficia a las soluciones IoT ya que los modelos de datos estandarizados se adaptan lo suficiente para permitir la portabilidad de datos para diferentes aplicaciones y estudios que ayuden a la implementación de Ciudades Inteligentes. El estándar que utilizan estos modelos de datos es NGSI.

El estándar NGSI tiene un gran numero diferentes clasificaciones de modelos de datos. Estos modelos de datos cuentan con los atributos necesarios para la representación de algún objeto de la vida real o un sensor.

Nuestro proyecto de tesis está enfocada a la movilidad urbana. La clasificación apropiada para nuestro proyecto es “Modelos de datos de armonizados de transporte”.

Modelos de datos armonizados de transporte

Estos modelos de datos describen principalmente entidades involucradas con aplicaciones inteligentes que se ocupan aspectos de transporte. Estos modelos han sido diseñados para ser lo más genéricos posible, lo que permite tratar con diferentes escenarios, tales como: control del flujo de tráfico, vehículos privados, vehículos públicos (autobuses, trenes, etc.), vehículos municipales (camiones de basura, unidades de limpieza, etc.), vehículos especiales (ambulancias, bomberos, etc.).

Lo modelos que encontramos en esta clasificación de modelos son:

a) *TrafficFlowObserved*: representa una observación sobre el flujo de tráfico.

b) *Road*: contiene una descripción geográfica y contextual armonizada de un camino.
c) *RoadSegment*: contiene una descripción geográfica y contextual armonizada de un segmento de carretera.
d) *Vehicle*: representa un vehículo con todas sus características individuales.
e) *VehicleModel*: representa un modelo de vehículo y captura sus propiedades estáticas, como dimensiones, materiales o características.

Modelo de datos para la representación de los Metrobuses de la CDMX

De acuerdo a la descripción de cada uno de los modelos que se ocupan en representar aspectos de transporte se identifica que el modelo *Vehicle* es el más apropiado para la representación de los Metrobuses. En la Tabla 4.1 se observan los atributos que son necesarios para la representación lógica de un vehículo de transporte público.

Tabla 4.1 Atributos para la representación del modelo Vehicle del estándar NGSI

Nombre del atributo	Descripción
id	Es el identificador de la entidad.
type	Indica el tipo de entidad que representa. En este caso *Vehicle*.
category	Indica la categoría en la que pertenece la entidad. En este caso *public*.
vehicleType	Indica el tipo de vehículo en la que pertenece la entidad. En este caso *bus*.
speed	Representa la velocidad que tiene el vehículo. Esta dato se requiere en la escala km/h.
mileageFromOdometer	Indica la distancia total en kilómetros recorrida por el vehículo desde su producción inicial.
heading	Representa la orientación de grados del vehículo. Son valores decimales donde 0 es norte y gira en sentido de las agujas del reloj.
location	Representa la ubicación del vehículo. Requiere los datos en un GeoJSON de tipo Point donde contiene los grados latitud y longitud.
servicesStatus	Representa el estado del vehículo.
dateObserved, dateModified	Indica el tiempo en que se ad-quiere la información en formato ISO 8601.
fleetVehicleId	Representa la identificación del vehículo en el contexto de una flota de vehículos.

Estos atributos son estructurados con la notación de la estructura JSON. En la Figura 4.19 se observa el JSON de la entidad *Vehicle*.

```
{
        "id": ""
        "type": "Vehicle",
    "dateObserved": {
            "type": "DateTime",
            "value":
    },
    "dateModified": {
            "type": "DateTime",
            "value":
    },
        "category": {
            "value": [
                "public"
            ]
        },
        "vehicleType": {
            "value": "bus"
        },
        "fleetVehicleId": {
            "value":
        },
        "location": {
            "type": "geo:json",
            "value": {
                "type": "Point",
                "coordinates": [,]
            },
            "metadata": {
                "timestamp": {
                    "type": "DateTime",
                    "value":
                }
            }
        },
        "serviceStatus": {
            "value":
        },
        "speed": {
            "value": speedVehicle,
            "metadata": {
                "timestamp": {
                    "type": "DateTime",
                    "value":
                }
            }
        },
        "heading": {
            "value": ,
            "metadata": {
                "timestamp": {
                    "type": "DateTime",
                    "value":
                }
            }
        },
        "mileageFromOdometer": {
            "value": ,
            "metadata": {
                "timestamp": {
                    "type": "DateTime",
                    "value":
                }
            }
        }
    }
```

Figura 4.19 Notación JSON para el modelo de datos *Vehicle*

Modelo de datos para la representación de la congestión vehicular de la CDMX

Para representar la congestión vehicular de cada uno de los segmentos de la Ciudad de México se identifica que el modelo apropiado es el denominado *TrafficFlowObserved*. En la Tabla 4.2 se observan los atributos que son necesarios para la representación lógica de la congestión vehiculas de un segmento de la Ciudad de México.

Tabla 4.2 Atributos para la representación del modelo *TrafficFlowObserved* del estándar NGSI

Nombre del atributo	Descripción
id	Es el identificador de la entidad.
type	Indica el tipo de entidad que representa. En este caso *Vehicle*.
occupancy	Representa la ocupación o factor de atasco del segmento.
addres.type	Indica el tipo de dirección. En este caso *StructuredValue*.
address.value.addressLocality	Representa el nombre de la localidad. En este caso "Ciudad de México".
address.value.addressCountry	Representa el nombre del país. En este caso "MX".
address.value.streetAddress	Representa el nombre de la dirección del segmento.
averageVehicleSpeed	Indica la velocidad media de tráfico vehicular.
location	Indica las coordenadas de los segmentos de carretera de los segmentos de la ciudad. Requiere los datos en un GeoJSON de tipo *MultiLineString* donde contiene los grados latitud y longitud.

Estos atributos son estructurados con la notación de la estructura JSON. En la Figura 4.20 se observa el JSON de la entidad *TrafficFlowObserved*.

```
{
    "id": "",
        "type": "TrafficFlowObserved",
        "dateObserved": {
            "type": "DateTime",
            "value":
        },
        "dateModified": {
            "type": "DateTime",
            "value":
        },
        "occupancy": {
            "value":
        },
        "address": {
            "type": "StructuredValue",
            "value": {
                "addressLocality": "Ciudad de México",
                "addressCountry": "MX",
                "streetAddress":
            }
        },
        "averageVehicleSpeed": {
            "value":
        },
        "location": {
            "type": "geo:json",
            "value": {
                "type": "MultiLineString",
                "coordinates": []
            }
        }
}
```

Figura 4.20 Notación JSON para el modelo de datos *TrafficFlowObserved*

4.4 Proceso 3: Creación de las reglas de mapeo de datos

En este proceso se diseñaron las reglas de mapeo para cada una de las estructuras de datos de origen: (1) Datos de los Metrobuses de la CDMX al modelo de datos *Vehicle*. (2) Datos de la congestión vehicular de la CDMX al modelo de datos *TrafficFlowObserved*. De igual forma, se diseñaron las funciones que permiten la transformación de los valores de atributos que lo requieran para poder ser aceptado por el modelo de datos. En la Figura 4.21 se pueden apreciar las tareas realizadas en este proceso.

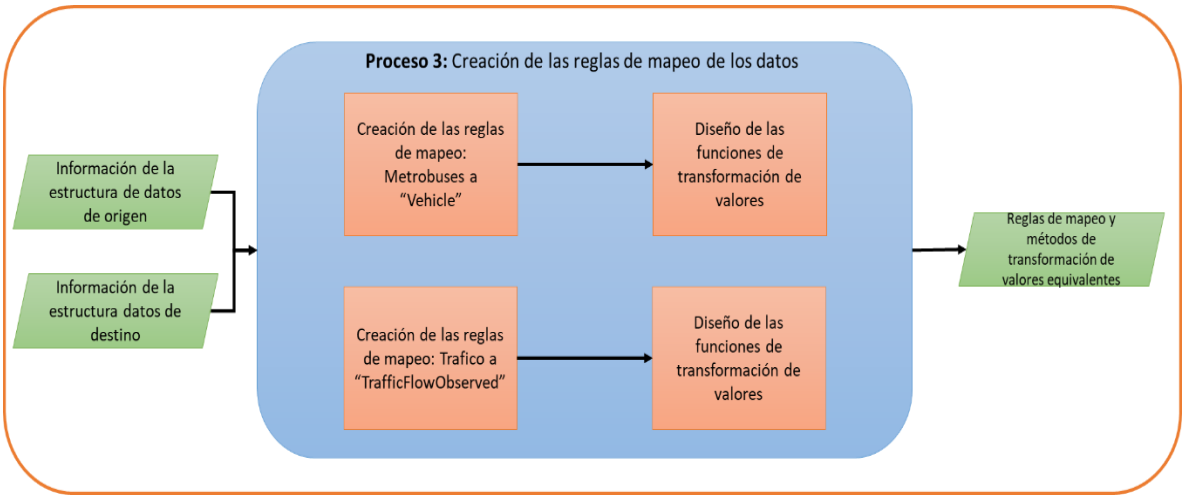


Figura 4.21 Proceso para la creación de las reglas de mapeo de los datos

4.4.1 Diseño de reglas de mapeo de los datos de los Metrobuses al modelo de datos *Vehicle*

En el diseño de las reglas de mapeo de los datos de los Metrobuses al modelo de datos *Vehicle* se diseñan reglas de mapeo y se diseñan las funciones de transformación de los valores de los atributos que lo necesiten.

Creación de las reglas de mapeo: Metrobuses al modelo *Vehicle*

En el contexto de nuestro proyecto de tesis, una regla de mapeo es encontrar el atributo equivalente entre el tipo de dato de origen y destino. Cabe mencionar que, no sé utilizan todos los valores de los datos de origen, solo se ocupan los necesarios para representar el modelo de datos *Vehicle*. En la Figura 4.22 se observan las reglas de mapeo de los datos de los Metrobuses al modelo de datos *Vehicle*.

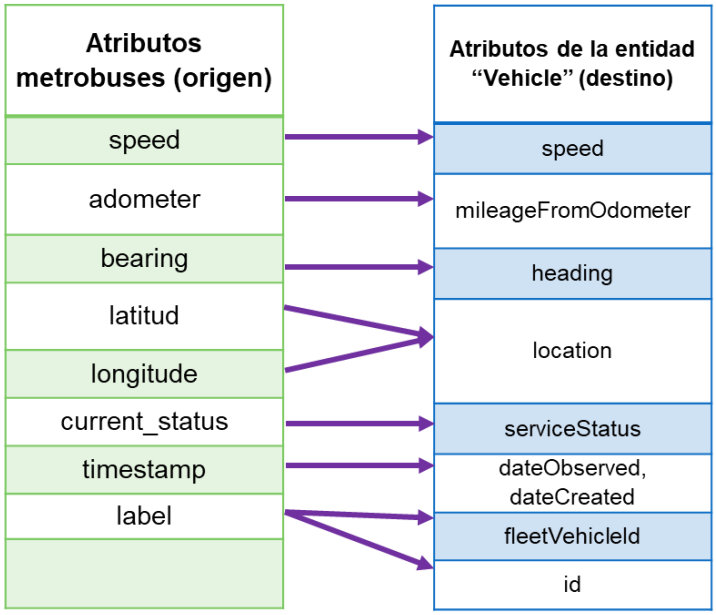


Figura 4.22 Reglas de mapeo de los datos de los Metrobuses al modelo *Vehicle*

Diseño de las funciones de trasformación de valores

Durante el mapeo es necesario transformar los valores de los atributos de origen a valores equivalentes según lo requiera el modelo de datos *Vehicle*. En la Figura 4.23 se observan las reglas de mapeo con las funciones que se diseñan para la trasformación.

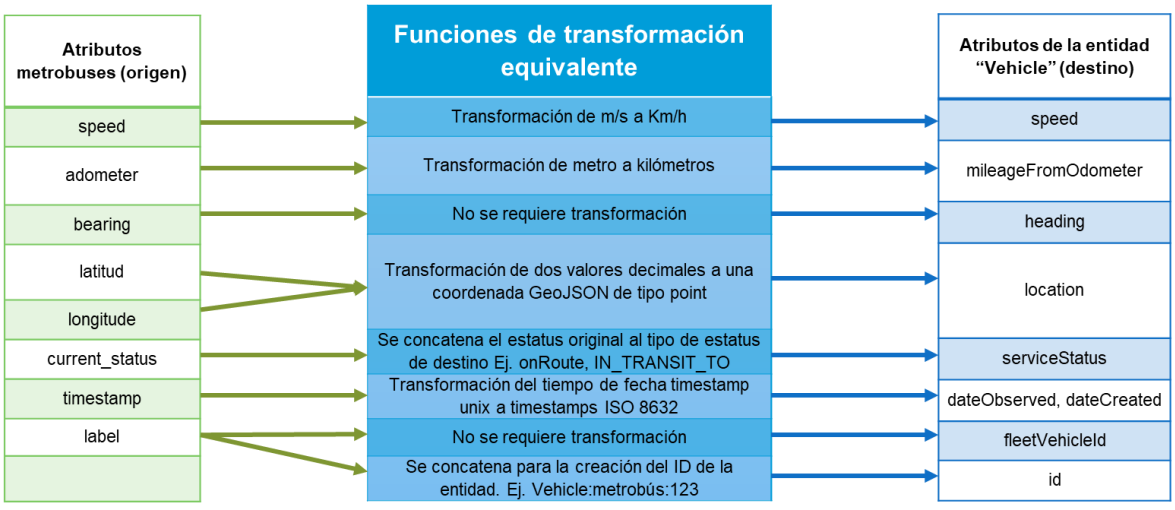


Figura 4.23 Funciones de trasformación de valores de los datos de los Metrobuses

4.4.2 Diseño de reglas de mapeo de los datos de la congestión vehicular al modelo de datos *TrafficFlowObserved*

En el diseño de las reglas de mapeo de los datos de la congestión vehicular al modelo de datos *TrafficFlowObserved* se diseñan reglas de mapeo y se diseñan las funciones de transformación de los valores de los atributos que lo necesiten.

Creación de las reglas de mapeo: Congestión vehicular al modelo *TrafficFlowObserved*

En el contexto de nuestro proyecto de tesis, una regla de mapeo es encontrar el atributo equivalente entre el tipo de dato de origen y destino. Cabe mencionar que, no sé utilizan todos los valores de los datos de origen, solo se ocupan los necesarios para representar el modelo de datos *TrafficFlowObserved*. En la Figura 4.24 se observan las reglas de mapeo de los datos de los Metrobuses al modelo de datos *TrafficFlowObserved*.

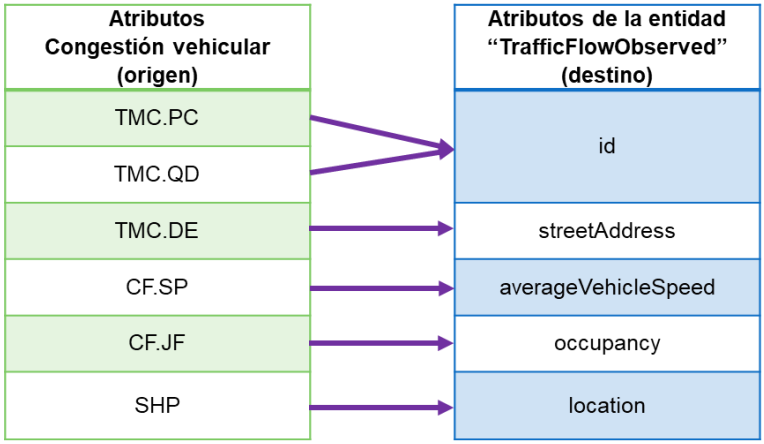


Figura 4.24 Reglas de mapeo de los datos de congestión vehicular al modelo *TrafficFlowObserved*

Diseño de las funciones de trasformación de valores

Durante mapeo es necesario transformar los valores de los atributos de origen a valores equivalentes según lo requiera el modelo de datos *TrafficFlowObserved*. En la Figura 4.27 se observan las reglas de mapeo con las funciones que se diseñan para la trasformación.

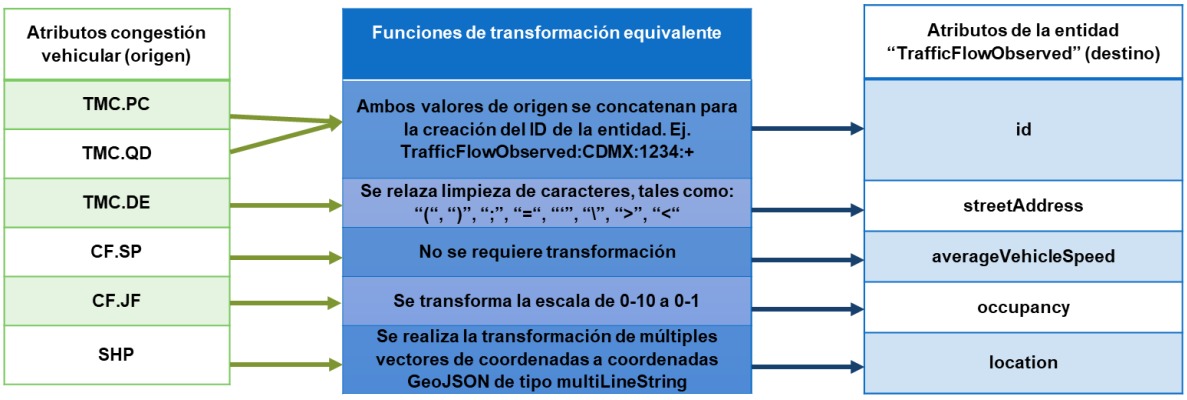


Figura 4.25 Funciones de trasformación de valores de los datos de los Metrobuses

4.5 Proceso 4: Desarrollo del espacio unificado de datos

En este proceso se implementa un servidor local con las funcionalidades de los componentes de FIWARE. Este servidor se encuentra localizado en el SITE de CENIDET y es considerado como el espacio unificado de datos. Asimismo, se desarrollan los alimentadores ADMU que se encargan de aplicar las reglas de mapeo y funciones de transformación que fueron diseñadas en el proceso 2.

Cabe mencionar que se desarrollan dos alimentadores ADMU independientes: (1) ADMU que extrae datos de los Metrobuses de la Ciudad de México, mapea los datos al modelo *Vehicle* y envía lo datos mapeados al servidor FIWARE. (2) ADMU que extrae los datos de la cogestión vehicular de la Ciudad de México provenientes de la Plataforma HEREMaps, mapea los datos al modelo *TrafficFlowObserved* y envía los datos mapeados al servidor FIWARE. En la Figura 4.26 se observa las tareas desarrollados durante este proceso.

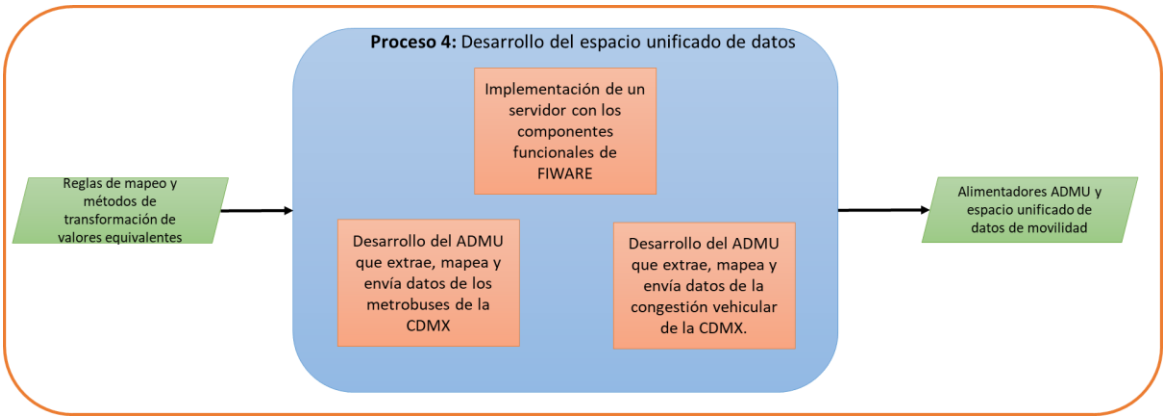


Figura 4.26 Proceso para el desarrollo del espacio unificado

En la Figura 4.27 se muestra un diagrama de bloques en el cual se observan los principales componentes que conforman cada uno de los alimentadores ADMU que realizan el trabajo de extraer, mapear y enviar los datos al servidor FIWARE.

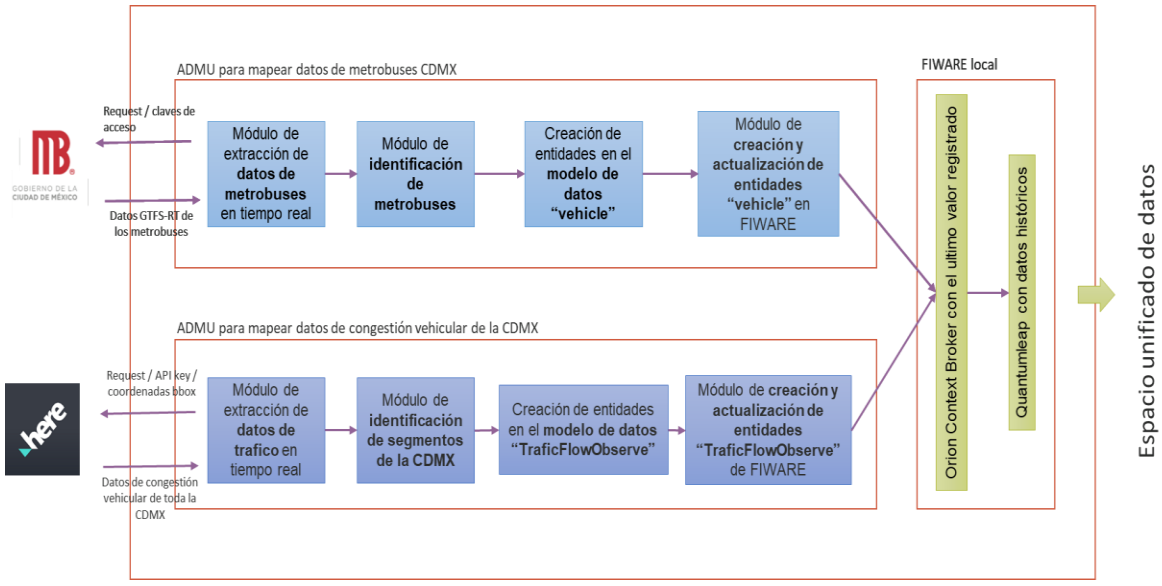


Figura 4.27 Diagrama de bloques de los ADMU´s para el desarrollo del espacio unificado

Cada uno de los ADMU´s son desarrollados con el lenguaje de programación Python debido a su gran variedad de bibliotecas y librerías que facilitan el tratamiento de grandes cantidades de datos.

4.5.1 Arquitectura IoT del flujo de los datos de movilidad

En esta sección se muestra una arquitectura IoT que consta de cuatro capas, tal y como se puede observar en Figura 4.28. Esta arquitectura muestra el flujo que tiene los datos desde su origen hasta ser alojados en nuestro espacio unificado.

Cabe mencionar que el flujo de los datos de la capa 1, la capa 2 y las bases de datos de la capa 3 son flujos transparentes para nuestro proyecto de tesis, ya que nuestros ADMU´s para mapear datos adquieren la información de las bases de datos de la capa 3.

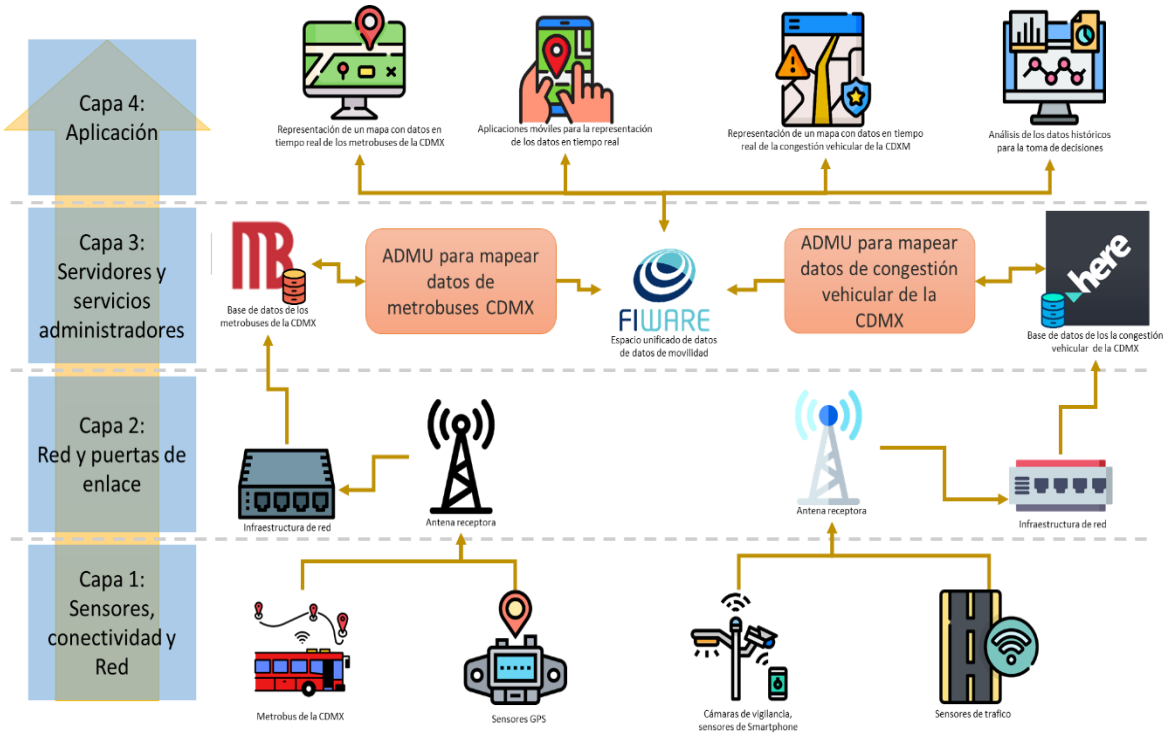


Figura 4.28 Arquitectura IoT de 4 capas

Capa 1: Sensores, conectividad y red

En esta capa se encuentran los diversos sensores que comparten información de sí mismos o de sus entornos. Puesto que nuestro proyecto de tesis utiliza datos de movilidad urbana, en la arquitectura se ejemplifican algunos sensores que comparte datos de la ubicación de los Metrobuses y congestión vehicular de la Ciudad de México.

Capa 2: Red y puertas de enlace

En esta capa se encuentra todos los dispositivos de comunicación de red que son necesarios para transferir los datos a una base de datos.

Capa 3: Servidores y servicios administradores

En esta capa se encuentran las distintas bases de datos donde los sensores alojan la información que están generando de sí mismos o de sus entornos. Estas bases de datos no utilizan modelos de datos genéricos que permiten la fácil reutilización de la información.

De igual forma, en esta capa se encuentran nuestros alimentadores ADMU propuestos en este proyecto de tesis. Estos alimentadores extraen datos de las diferentes bases de datos y los transforman al modelo de datos del estándar NGSI. Estos datos son alojados en nuestro espacio unificado que está construido con las funcionalidades de FIWARE.

Este espacio unificado utiliza modelos de datos genéricos que facilitan el entendimiento y la reutilización de los datos.

Capa 4: Aplicación

En esta capa se encuentran algunas de las muchas aplicaciones que tienen los datos estandarizados de nuestro espacio unificado de datos. En este proyecto de tesis se proponen una seria de aplicaciones web que representan en tiempo real la ubicación de los Metrobuses o congestión vehicular de la Ciudad de México dentro de un mapa.

Asimismo, se pueden generar análisis estadísticos con los datos históricos que se encuentran alojados en Quantumleap para tomar las mejores decisiones que ayuden a la mejora de la movilidad urbana.

Cabe mencionar que todas las aplicaciones y análisis se realizan con los datos estandarizados que se encuentran alojados dentro de nuestro espacio unificado de datos. Y, puesto que el espacio unificado de datos está construido con componentes de FIWARE es relativamente sencillo generar este tipo de aplicaciones y análisis. Ya que nuestro espacio unificado brinda APIs de acceso rápido a datos.

4.5.2 ADMU para mapear datos de Metrobuses CDMX

El prototipo del Alimentador de Datos de Movilidad Urbana para mapear datos de los Metrobuses de la Ciudad de México se encuentra organizada por cuatro componentes principales (Figura 4.29).

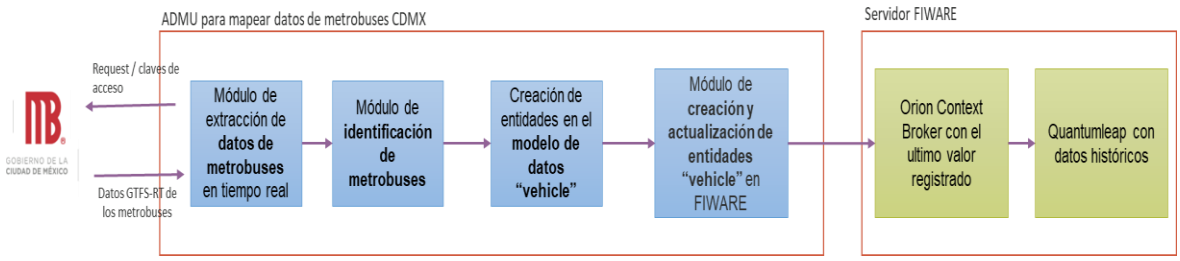


Figura 4.29 ADMU para mapear datos de los Metrobuses de la CDMX

En seguida se describen cada uno de los componentes, así como su desarrollo.

Módulo de extracción de datos de los Metrobuses en tiempo real

Este módulo se encarga de realizar las peticiones REST al sitio donde se encuentran alojados los datos en tiempo real de los Metrobuses de la Ciudad de México. La extracción de los datos es de forma masiva, es decir, obtiene los datos de la ubicación en tiempo real de todos los Metrobuses que se encuentran compartiendo su información.

En términos generales, este módulo obtiene una lista con la información en tiempo real de la ubicación de los Metrobuses de la Ciudad de México. En la Figura 4.30 se puede observar la función que se encarga de obtener los datos de todos los Metrobuses.

```python
#Paqueterias de python para utilizar datos GTFS-RT y request
from google.transit import gtfs_realtime_pb2
import requests

def consumeGTFS():
    feed = gtfs_realtime_pb2.FeedMessage()
    # peticion request hacia la liga http://app.citi-mb.mx/GTFS-RT/vehiculosPosicion
    user='MB456E;V'
    pss='NCKQONZRLW'
    response = requests.get('http://'+user+':'+pss+'@app.citi-mb.mx/GTFS-RT/vehiculosPosicion')
    feed.ParseFromString(response.content)

    return feed
```

Figura 4.30 Función para extraer datos de los Metrobuses

Módulo de identificación de los Metrobuses

Los datos de todos los Metrobuses se almacenan en forma masiva en una lista por lo que fue necesario desarrollar un módulo que sea el responsable de recorrer toda la lista e identificar a los Metrobuses de forma individual. En esta identificación de los Metrobuses se extraen los datos que son necesarios para la representación de la entidad *Vehicle*. Asimismo, se aplican las funciones de trasformación de valores acorde a lo requerido por el estándar NGSI de FIWARE para la representación de entidades.

En la Figura 4.31 se muestra el módulo desarrollado para identificar y extraer los atributos de cada uno de los Metrobuses.

```python
    #Datos GTFS-RT dentro del Feed (datos de todos los metrobuses)
    feed=consumeGTFS()
    #Unicia el contador el cual recorre cada entidad dentro de GTFS-RT
    i=0
    #Acceso a los datos por vehiculo, conservando el valor de los atributos que se necesitan
    for i in range(len(feed.entity)):
        metroBus=feed.entity[i]
        idVehicle=metroBus.vehicle.vehicle.id
        labelVehicle=metroBus.vehicle.vehicle.label
        latitudeVehicle=metroBus.vehicle.position.latitude
        longitudeVehicle=metroBus.vehicle.position.longitude
        #tranformacion del dato TIMESTAMP
        timeConsulta=datetime.fromtimestamp(metroBus.vehicle.timestamp).isoformat()
        #transformacion del dato CURRENT_STATUS
        if (metroBus.vehicle.current_status == 2):
            statusVehicle="onRute,IN_TRANSIT_TO"
        else:
            statusVehicle="onRute,STOPPED_AT"
        #transformacion del dato SPEED
        speedVehicle=metroBus.vehicle.position.speed*3.6
        bearingVehicle=metroBus.vehicle.position.bearing
        #tranformacion del dato ODOMETER
        odometerVehicle=metroBus.vehicle.position.odometer/1000
```

Figura 4.31 Módulo para identificar a los Metrobuses

En donde la línea número 28 invoca la función que consume los datos de los Metrobuses y son guardados en la variable “feed”. Esta variable es ingresada a un ciclo *for* para poder recorrer toda la lista de los Metrobuses. Durante este recorrido, se adquieren y transforman los valores de los atributos que son necesarios para representar la entidad *Vehicle* del estándar NGSI.

Creación de entidades en el modelo de datos *Vehicle*

Este componente crea las entidades *Vehicle* con los datos de cada uno de los Metrobuses. Estos datos fueron previamente transformados al estándar NGSI. Para lograr la creación de las entidades son aplicadas las reglas de mapeo previamente diseñadas.

Las Figuras Figura 4.32 - Figura 4.33 muestran las entidades *Vehicle* con ayuda de la notación de objetos JSON.

```
data={
    "id": "Vehicle:MetroBusCDMX:"+labelVehicle,
    "type": "Vehicle",
    "dateObserved": {
        "type": "DateTime",
        "value": str(timeConsulta)
    },
"dateModified": {
        "type": "DateTime",
        "value": str(timeConsulta)
},
    "category": {
        "value": [
            "public"
        ]
    },
    "vehicleType": {
        "value": "bus"
    },
    "fleetVehicleId": {
        "value": labelVehicle
    },
    "location": {
        "type": "geo:json",
        "value": {
            "type": "Point",
            "coordinates": [
                longitudeVehicle,
                latitudeVehicle
            ]
        },
        "metadata": {
            "timestamp": {
                "type": "DateTime",
                "value": timeConsulta
            }
        }
    },
    "serviceStatus": {
        "value": str(statusVehicle)
    },
    "speed": {
        "value": speedVehicle,
        "metadata": {
            "timestamp": {
                "type": "DateTime",
                "value": timeConsulta
            }
        }
    },
    "heading": {
        "value": bearingVehicle,
        "metadata": {
            "timestamp": {
                "type": "DateTime",
                "value": timeConsulta
            }
        }
    },
    "mileageFromOdometer": {
        "value": odometerVehicle,
        "metadata": {
            "timestamp": {
                "type": "DateTime",
                "value": timeConsulta
            }
        }
    }
}
```

Figura 4.32 JSON para la creación de la entidad *Vehicle*

```
data={
"dateObserved": {
        "type": "DateTime",
        "value": timeConsulta
    },
    "dateModified": {
        "type": "DateTime",
        "value": timeConsulta
    },
    "location": {
        "type": "geo:json",
        "value": {
            "type": "Point",
            "coordinates": [
                longitudeVehicle,
                latitudeVehicle]
        },
        "metadata": {
            "timestamp": {
                "type": "DateTime",
                "value": timeConsulta
            }
        }
    },
    "serviceStatus": {
        "value": statusVehicle
    },
    "speed": {
        "value": speedVehicle,
        "metadata": {
            "timestamp": {
                "type": "DateTime",
                "value": timeConsulta
            }
        }
    },
    "heading": {
        "value": bearingVehicle,
        "metadata": {
            "timestamp": {
                "type": "DateTime",
                "value": timeConsulta
            }
        }
    },##Se agrega odometro
    "mileageFromOdometer": {
        "value": odometerVehicle,
        "metadata": {
            "timestamp": {
                "type": "DateTime",
                "value": timeConsulta
            }
        }
    }
}
```

Figura 4.33 JSON para la actualización de la entidad *Vehicle*

Cabe mencionar que se crean dos tipos de objetos JSON de la entidad *Vehicle*: (1) Un JSON que se utiliza para la creación de las entidades. (2) Un JSON para la actualización de las entidades. Cada uno de estas estructuras JSON son invocados mediante funciones que espera los datos de los Metrobuses identificados y transformados como parámetros de entrada.

Módulo de creación y actualización de entidades *Vehicle* en FIWARE

Este módulo se encarga de enviar las entidades *Vehicle* al servidor FIWARE una a la vez, específicamente al *Orion Context Broker*. Para que este módulo pueda crear de forma correcta las entidades dentro del servidor FIWARE es necesario enviar mediante petición POST la suscripción del modelo de datos *Vehicle* al OCB. La suscripción se puede observar en la Figura 4.34.

La liga donde se envía la suscripción es: http://ipServer/v2/subscriptions/

```
{
    "description":"Notificación Quantumleap para Vehiculo",
    "subject":{
        "entities":[{
            "idPattern":"Vehicle:MetroBusCDMX:*",
            "type":"Vehicle"
        }
        ],
        "condition":{
            "attrs":[
                "dateModified"
            ]
        }
    },
    "notification":{
        "attrs":[
            "location",
            "serviceStatus",
            "speed",
            "heading",
            "mileageFromOdometer",
            "dateModified",
            "dateObserved"
        ],
        "http":{
            "url":"http://quantumleap:8668/v2/notify"
        },
        "metadata":[
            "dateCreated",
            "dateModified"
        ]
    }
}
```

Figura 4.34 Suscripción de la entidad *Vehicle*

Con el objetivo de alojar los datos de las entidades en el *Orion Context Broker* se utilizan peticiones REST, específicamente, la petición POST para crear entidades y la petición PATCH para actualizarlas. Cada petición tiene respuestas numéricas de la solicitud, algunas de ellas son:

- Respuesta 201 para la petición POST: creación de la entidad de forma correcta.
- Respuesta 422 para la petición POST: entidad no creada, la entidad ya existe.
- Respuesta 204 para la petición PACH: actualización de la entidad de forma correcta.

Tomando en cuenta las respuestas numéricas de las peticiones REST se puede identificar en que momento es necesario actualizar una entidad. La lógica de este módulo establece lo siguiente:

"Crea la entidad del Metrobús en turno en el Orion Context Broker, si la respuesta de la petición 422, no crear la entidad, solo actualizarla con los datos del Metrobús en turno"

La liga para crear entidades *Vehicle* es: http://ipServer:1026/v2/entities/

Liga para actualizar la entidad *Vehicle* es: http://ipServer:1026/v2/entities/**idEntidad**/attrs

El valor de "idEntidad" es el id de la entidad la cual será actualizada.

En la Figura 4.35 se puede observar el código para crear y actualizar entidades en el servidor FIWARE donde se observa el llamado de las funciones para llenar los JSONs con los datos de los Metrobuses y al mismo tiempo enviar las entidades el servidor FIWARE.

```
#URL para creación de entidades
urlCreacionEntidad = "http://"+ str(ipServer) +":1026/v2/entities/"
payloadCreacionEntidad=llenaJSONcreacionEntidad(labelVehicle,
                                                latitudeVehicle,
                                                longitudeVehicle,
                                                timeConsulta,
                                                statusVehicle,
                                                speedVehicle,
                                                bearingVehicle,
                                                odometerVehicle)

estatusResquestPOST = requests.post(url= urlCreacionEntidad,
                                    data=payloadCreacionEntidad,
                                    headers= headers_string)
#imprime el estado del POST
print("VEHICLE Estado POST: {}" .format(estatusResquestPOST.status_code))

if(estatusResquestPOST.status_code==422):
    #URL para actualización de entidades
    urlActualizacionEntidad = "http://"+ str(ipServer) +":1026/v2/entities/Vehicle:MetroBusCDMX:"+str(labelVehicle)+"/attrs"
    payloadActualizacionEntidad=llenaJSONactualizacion(latitudeVehicle,
                                                       longitudeVehicle,
                                                       timeConsulta,
                                                       statusVehicle,
                                                       speedVehicle,
                                                       bearingVehicle,
                                                       odometerVehicle)
    estatusResquestPACH = requests.patch(url= urlActualizacionEntidad,
                                         data=payloadActualizacionEntidad,
                                         headers= headers_string)
    #imprime el estado del PACH
    print("VEHICLE Estado PACH: {}" .format(estatusResquestPACH.status_code))
```

Figura 4.35 Código para crear o actualizar entidades de *Vehicle* en FIWARE

Los datos que son actualizados no son borrados por el OCB., ya que, de forma transparente, el OCB envía los datos al componente Quantumleap para generar datos históricos de cada una de las entidades. Este ADMU se automatiza para su ejecución cada 60 segundos. Este tiempo es el que se tiene para extraer datos totalmente actualizados.

El código completo para el ADMU que utiliza datos de los Metrobuses de la CDMX se encuentra en el anexo B denominado Alimentadores ADMU desarrollados.

4.5.3 ADMU para mapear datos de la congestión vehicular de la CDMX

El prototipo del Alimentador de Datos de Movilidad Urbana para mapear datos de la congestión vehicular de la Ciudad de México se encuentra organizada por cuatro componentes principales (Figura 4.36).

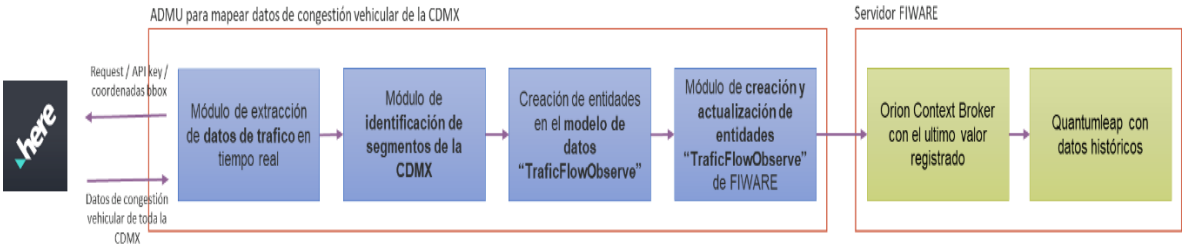


Figura 4.36 ADMU para mapear datos de congestión vehicular de la CDMX

Módulo de extracción de datos de la congestión vehicular en tiempo real

Este módulo se encarga de realizar las peticiones REST al API donde se encuentran alojados los datos en tiempo real de la congestión vehicular de la Ciudad de México. La extracción de los datos es de forma masiva, es decir, obtiene los datos de la congestión vehicular de todos los segmentos que conforman a la Ciudad de México.

Cabe mencionar que no existen una correspondencia 1 a 1 entre un segmento y una calle, por ejemplo, una calle larga, como es el caso de la Avenida Insurgentes se divide en múltiples segmentos. Existen calles pequeñas que si pueden ser especificadas con un solo segmento.

En términos generales, este módulo obtiene una lista con la información en tiempo real de la congestión vehicular de los segmentos de la Ciudad de México. En la Figura 4.37 se puede observar la función que se encarga de obtener los datos de todos los segmentos de la Ciudad de México.

```
7   import requests
8
9   def consumeHereMaps():
10      coordenadasBbox = '19.725181,-99.316750;19.061956,-98.788376'
11      apiKey= 'dvK17Bzm7_rtFgRUIKH_BvZOcOdmrBDC0H7GiapjlWE'
12
13      url = 'https://traffic.ls.hereapi.com/traffic/6.2/flow.json?bbox='+ str(coordenadasBbox) +'&responseattributes=sh&apiKey='+str(apiKey)
14      dataHere = requests.get(url)
15      dataHereJson = dataHere.json()
16
17      return dataHereJson
```

Figura 4.37 Función para extraer datos de congestión vehicular de la CDMX

Módulo de identificación de los Metrobuses

Los datos de todos los segmentos de la CDMX se extraen de forma masiva y almacenados en una lista por lo cual fue necesario desarrollar un módulo que se encarga de recorrer toda la lista e identificar a los segmentos de la Ciudad de México de forma individual.

Durante esta identificación de los segmentos de la ciudad, se extraen los datos que son necesarios para la representación de la entidad *TrafficFlowObserved*. Asimismo, se aplican las funciones de trasformación de valores acorde a lo requerido por el estándar NGSI.

En la Figura 4.38 se puede apreciar el módulo de software que se encarga de identificar y extraer los atributos de cada uno de los segmentos de la Ciudad de México.

```
23          #Función que consulta los datos de HERE Maps
24          dataHereJson=consumeHereMaps()
25          #Recorrido que se hace a todo el JSON
26          for RWS in range(len(dataHereJson['RWS'])):
27            for RW in range(len(dataHereJson['RWS'][RWS]['RW'])):
28              for FIS in range(len(dataHereJson['RWS'][RWS]['RW'][RW]["FIS"])):
29                for FI in range(len(dataHereJson['RWS'][RWS]['RW'][RW]["FIS"][FIS]['FI'])):
30                  datos = dataHereJson['RWS'][RWS]['RW'][RW]['FIS'][FIS]['FI'][FI]
31                  TMC_PC=datos['TMC']['PC']
32                  TMC_QD=datos['TMC']['QD']
33                  #Limpia el texto de los simbolos especificos
34                  TMC_DE=limpiaTexto(datos['TMC']['DE'])
35                  CF_SP=datos['CF'][0]['SP']
36                  #Re-escala el valor
37                  CF_JF=rescale(float(datos['CF'][0]['JF']),0,10,0,1)
38                  #Tranforma el conjunto de coordenadas en un geoJSON MultiLineString
39                  SHP=getCoordinates(datos['SHP'])
40                  horaConsulta=cambiaZonaHoraria(dataHereJson['CREATED_TIMESTAMP'])
```

Figura 4.38 Módulo para identificar los segmentos de la CDMX

La línea número 24 invoca la función que consume los datos de congestión vehicular y son guardados en la variable “dataHereJson”. Esta variable es ingresada a una serie de ciclos *for* para poder recorrer toda la lista de los segmentos de la Ciudad de México. Durante este recorrido, se adquieren y transforman los valores de los atributos que son necesarios para representar la entandad *TrafficFlowObserved* del estándar NGSI.

Creación de entidades en el modelo de datos *TrafficFlowObserved*

Este componente crea las entidades *TrafficFlowObserved* con los datos de un segmento de la ciudad a la vez, estos datos fueron previamente transformados. Para lograr la creación de las entidades son aplicadas las reglas de mapeo previamente diseñadas.

Las Figuras Figura 4.39 - Figura 4.40 las entidades *TrafficFlowObserved* con ayuda de la notación de objetos JSON.

```
data={
    "id": "TrafficFlowObserved:CDMX:"+str(TMC_PC)+str(TMC_QD),
          "type": "TrafficFlowObserved",
          "dateObserved": {
              "type": "DateTime",
              "value": str(horaConsulta)
          },
          "dateModified": {
              "type": "DateTime",
              "value": str(horaConsulta)
          },
          "occupancy": {
              "value": float(CF_JF)
          },
          "address": {
              "type": "StructuredValue",
              "value": {
                  "addressLocality": "Ciudad de México",
                  "addressCountry": "MX",
                  "streetAddress": str(TMC_DE)
              }
          },
          "averageVehicleSpeed": {
              "value": float(CF_SP)
          },
          "location": {
              "type": "geo:json",
              "value": {
                  "type": "MultiLineString",
                  "coordinates": SHP
              }
          }
    }
```

Figura 4.39 JSON para la creación de la entidad *TrafficFlowObserved*

```
data={
        "dateObserved": {
            "type": "DateTime",
            "value": horaConsulta
        },
        "dateModified": {
            "type": "DateTime",
            "value": horaConsulta
        },
        "occupancy": {
            "value": float(CF_JF)
        },
        "averageVehicleSpeed": {
            "value": float(CF_SP)
        }
    }
```

Figura 4.40 JSON para la actualización de la entandad *TrafficFlowObserved*

Al igual que en el caso de la información de los Metrobuses, se crean dos tipos de objetos JSON, uno para la creación de entidades y otro para la actualización de las mismas.

Módulo de creación y actualización de entidades *TrafficFlowObserved* en FIWARE

Este módulo se encarga de enviar las entidades *TrafficFlowObserved* al servidor FIWARE una a la vez, específicamente al *Orion Context Broker*. Para que este módulo pueda crear de forma correcta las entidades dentro del servidor FIWARE es necesario mediante petición POST la suscripción del modelo de datos *Vehicle* al OCB. La suscripción de puede observar en la Figura 4.41.

La liga donde se envía la suscripción es: http://ipServer/v2/ subscriptions/

```
{
    "description":"Notificación Quantumleap para Trafico",
    "subject":{
        "entities":[{
            "idPattern":"TrafficFlowObserved:*",
            "type":"TrafficFlowObserved"
        }
        ],
        "condition":{
            "attrs":[
                "dateModified"
            ]
        }
    },
    "notification":{
        "attrs":[
            "dateObserved",
            "dateModified",
            "occupancy",
            "averageVehicleSpeed"
        ],
        "http":{
            "url":"http://quantumleap:8668/v2/notify"
        },
        "metadata":[
            "dateCreated",
            "dateModified"
        ]
    }
}
```

Figura 4.41 Suscripción de la entidad *TrafficFlowObserved*

Para el proceso de creación o actualización de las entidades del tipo *TrafficFlowObserved* se sigue las mismas reglas utilizadas para el caso de la información de los Metrobuses de la CDMX.

En la Figura 4.42 se puede observar el código para crear y actualizar entidades en el servidor FIWARE donde se puede apreciar el llamado de las funciones para llenar los JSONs con los datos de congestión vehicular de los segmentos de la Ciudad de México y al mismo tiempo enviar las entidades al servidor FIWARE.

```
#URL para creación de entidades
urlCreacionEntidadTrafico = "http://"+ str(ipAddressServer) +":1026/v2/entities/"
payloadCreacionEntidadTrafico=llenaJSONcreacionEntidadTrafico(TMC_PC,
                                                              TMC_QD,
                                                              TMC_DE,
                                                              CF_SP,
                                                              CF_JF,
                                                              SHP,
                                                              horaConsulta)

estatusResquestPOSTTrafico = requests.post(url= urlCreacionEntidadTrafico,
                                           data=payloadCreacionEntidadTrafico,
                                           headers= headers_string)

#imprime el estado del POST
print("TRAFFIC Estado POST: {}" .format(estatusResquestPOSTTrafico.status_code))

if(estatusResquestPOSTTrafico.status_code==422):
    #URL para actualización de entidades
    urlActualizacionEntidadTrafico = "http://"+ str(ipAddressServer) +":1026/v2/entities/TrafficFlowObserved:CDMX:"+str(TMC_PC)+str(TMC_QD)+"/attrs"
    payloadActualizacionEntidadTrafico = llenaJSONactualizacionEntidadTrafico(CF_SP,
                                                                              CF_JF,
                                                                              horaConsulta)
    estatusResquestPACHTrafico = requests.patch(url= urlActualizacionEntidadTrafico,
                                                data=payloadActualizacionEntidadTrafico,
                                                headers= headers_string)
    #imprime el estado del PACH
    print("TRAFFIC Estado PACH: {}" .format(estatusResquestPACHTrafico.status_code))
```

Figura 4.42 Código para crear o actualizar entidades *TrafficFlowObserved* en FIWARE

Este ADMU se automatiza para su ejecución cada 5 minutos. Este tiempo es el que se tiene para extraer datos totalmente actualizados. El código completo para el ADMU que utiliza datos de la congestión vehicular de la CDMX se encuentra en el anexo B denominado Alimentadores ADMU desarrollados.

Capítulo 5

Aplicaciones desarrolladas para el consumo de datos del espacio unificado

5.1 Demostración de aplicaciones implementadas con los datos del espacio unificado

En este capítulo se muestran una serie de aplicaciones que se pueden desarrollar a partir de los datos de movilidad urbana de nuestro espacio unificado de datos. Las aplicaciones fueron desarrolladas en un ambiente controlado dentro de una red local. Las herramientas utilizadas fueron *JavaScript*, HTML y el API de LeafLet para la representación del mapa.

5.1.1 Desarrollo y demostración de las aplicaciones web para la representación de los Metrobuses en tiempo real y datos históricos de los Metrobuses y congestión vehicular de la CDMX

Se desarrollaron varias aplicaciones web las cuales adquieren información en tiempo real o datos históricos de los Metrobuses y congestión vehicular de la Ciudad de México y son representados en un mapa. Estos mapas comprueban la extracción de los datos en tiempo real y que se guardan datos históricos de forma correcta.

Los datos que se representan en estas aplicaciones de mapa son extraídos del espacio unificado de datos propuesto en este proyecto de tesis. Esta adquisición se logra mediante peticiones *REST*, específicamente la petición *GET*.

Esta petición tiene respuestas numéricas, tales como:

- Repuesta *Request timed out*: el servidor al cual se le hace la petición no contesta.
- Respuesta 400: no se encuentra la URL solicitada.
- Respuesta 200: petición *GET* bien realizada. Adicional a esta respuesta, en nuestro caso devuelve el JSON con las entidades solicitadas.

Una de las funcionalidades de la Plataforma FIWARE es que brinda APIs de acceso rápido a datos. Estas APIs nos permites extraer información en tiempo real que se encuentra alojada en el OCB o información histórica que se encuentra en el Quantumleap.

API de acceso a datos en tiempo real

EL API de FIWARE brinda accesibilidad para poder filtrar la información desde la liga del API.

A continuación, se muestra unos ejemplos que se utilizaron las aplicaciones web.

Filtro para obtener datos en tiempo real de un solo Metrobús en específico

http://ipServer:1026/v2/entities/vehicle:Metrobus:**noUnidad**

en donde el “noUNidad” es el número económico del Metrobús.

Filtro para obtener datos en tiempo real de los 10 primeros Metrobuses

http://ipServer:1026/v2/entities?type=Vehicle&limit=10

Filtro para obtener datos en tiempo real de todos los Metrobuses

http://ipServer:1026/v2/entities?type=Vehicle&limit=1000

en donde el valor de *limit* supera el número de unidades. Como resultado se obtienen los datos de los 852 Metrobuses.

Filtro para obtener datos en tiempo real de la congestión vehicular un solo segmento de la ciudad

http://ipServer:1026/v2/entities/*TrafficFlowObserved*:CDMX:**idSegmentoySentido**

en donde "idSegmentoySentido" es el id y el sentido del segmento que se quiere obtener su información. Ejemplo de este valor: 1234+.

Filtro para obtener datos en tiempo real de la congestión vehicular de los 10 primeros segmentos de la ciudad

http://ipServer:1026/v2/entities?type=*TrafficFlowObserved*&limit=10

Filtro para obtener datos en tiempo real de la congestión vehicular de todos los segmentos de la ciudad.

Cabe mencionar que por cada filtro puede mostrar un máximo de 1,000 entidades. Y puesto que son más de 4,000 segmentos de la Ciudad de México se generaron una serie de filtros para obtener todos los segmentos.

1) http://ipServer:1026/v2/entities?type=*TrafficFlowObserved*&limit=1000&offset=0
2) http://ipServer:1026/v2/entities?type=*TrafficFlowObserved*&limit=1000&offset=1000
3) http://ipServer:1026/v2/entities?type=*TrafficFlowObserved*&limit=1000&offset=2000
4) http://ipServer:1026/v2/entities?type=*TrafficFlowObserved*&limit=1000&offset=3000
5) http://ipServer:1026/v2/entities?type=*TrafficFlowObserved*&limit=1000&offset=4000

En donde la variable *offset* indica el número de entidad donde va a comenzar a obtener la información, mientras que la variable *limit* indica el número de entidades que se desean obtener. Ejemplos, en el ejemplo 1 se obtiene datos de 1,000 entidades a partir de la entidad 0. En el ejemplo 3 se obtienen datos de 1,000 a partir de la entidad número 2,000.

La información obtenida por cada una de los filtros fue representada en una aplicación web de un mapa. Esta aplicación es automatizada para la adquisición y representación de los datos. Esta actualización se ejecuta cada 60 segundos para los datos de los Metrobuses y cada 5 para los datos de congestión vehicular.

API de acceso a datos históricos

Filtro de acceso a los datos históricos de un Metrobús

http://ipServer:8668/v2/entities/Vehicle:MetrobúsCDMX:**noUnidad**?fromDate=**timepoInicio**&toDate=**tiempoFin**

en donde el "noUnidad" es el número económico del Metrobús, la "tiempoInicio" y "tiempoFin" son las fechas y horas donde inicia el monitoreo y donde termina respectivamente.

Filtro de acceso a datos históricos de la congestión vehicular de un segmento de la ciudad

http://ipServer:8668/v2/entities/*TrafficFlowObserved*:CDMX:
idSegmentoySentido?fromDate=**timepoInicio**&toDate=**tiempoFin**

en donde "idSegmentoySentido" es el id y el sentido del segmento que se quiere obtener su información. Ejemplo de este valor: 1234+., la "tiempoInicio" y "tiempoFin" son las fechas y horas donde inicia el monitoreo y donde termina respectivamente.

Cabe mencionar que las fecha en ambos filtros debe de tener el siguiente formato: YYYY-MM-DDTHH:mm:SS en donde YYYY es el año a cuatro dígitos numéricos, MM el mes a dos dígitos numéricos, DD el día a dos dígitos numéricos, T un separador, HH la hora a dos dígitos numéricos, mm los minutos a dos dígitos numéricos y SS los segundos a dos dígitos numéricos.

Representación de los datos del espacio unificado en un Mapa

Los datos de movilidad urbana almacenados en nuestro espacio unificado son representados en un mapa. Para su representación se desarrollaron cuatro aplicaciones web demos: (1) Mapa con datos de los Metrobuses en tiempo real. (2) Mapa con datos de congestión vehicular en tiempo real. (3) Mapa con datos históricos de un solo Metrobús. (4) Mapa con datos históricos de congestión vehicular de un solo segmento de la ciudad.

Las aplicaciones web son desarrolladas con las herramientas HTML, *JavaScript* y se utiliza el API de leaflet para la representación del mapa.

Mapa con datos de los Metrobuses en tiempo real

Para la extracción de los datos en tiempo real de todos los Metrobuses de la Ciudad de México se desarrolla la siguiente liga.

http://ipServer:1026/v2/entities?type=Vehicle&limit=100&offset=0

Con esta liga se desarrolla una función *JavaScript* para poder obtener los datos en tiempo real y representarlos en un mapa. La Figura 5.1 es la función que desempeña esta actividad.

```
35      //Consulta de coordenadas y dibujo de calles
36      function pintaMetrobuses(){
37          $(".leaflet-marker-icon").remove();
38          $.ajax({
39              //url: "http://fiwareCenidet:1026/v2/entities?type=Vehicle&limit=100&offset=100",
40              //url: "http://fiwareCenidet:1026/v2/entities?type=Vehicle&limit=1000&offset=0&idPattern=^Vehicle:MetroBusCDMX:11",
41              url: "http://172.16.10.220:1026/v2/entities?type=Vehicle&limit=100&offset=0",
42              type: "GET",
43              dateType: "json",
44          }).done(function(data){
45              $.each(data, function(key, datos){
46                  //console.log(datos)
47                  coordenadasMetrobus = datos.location.value.coordinates
48                  console.log(coordenadasMetrobus)
49
50                  var marker = L.marker([coordenadasMetrobus[1], coordenadasMetrobus[0]], {icon: customIcon, title:"Metrobus número: " + datos.fleetVehicleId.value + "\
51
52                  marker.addTo(map);
53                  //L.geoJSON(coordenadasMetrobus, {onEachFeature: onEachFeature}).addTo(map);
54              })
55
56          })
57      }
58      setInterval('pintaMetrobuses()',15000);
```

Figura 5.1 Función *JavaScript* que representa todos los Metrobuses en un mapa

La línea número 50 extrae las coordenadas para la representación en el mapa, asimismo, extraer el resto de los valores de los atributos para que puedan ser representados en un cuadro de información.

La línea número 58 ejecuta la función cada 15 segundos.

En la Figura 5.2 se puede apreciar la representación de todos los Metrobuses de la Ciudad de México.

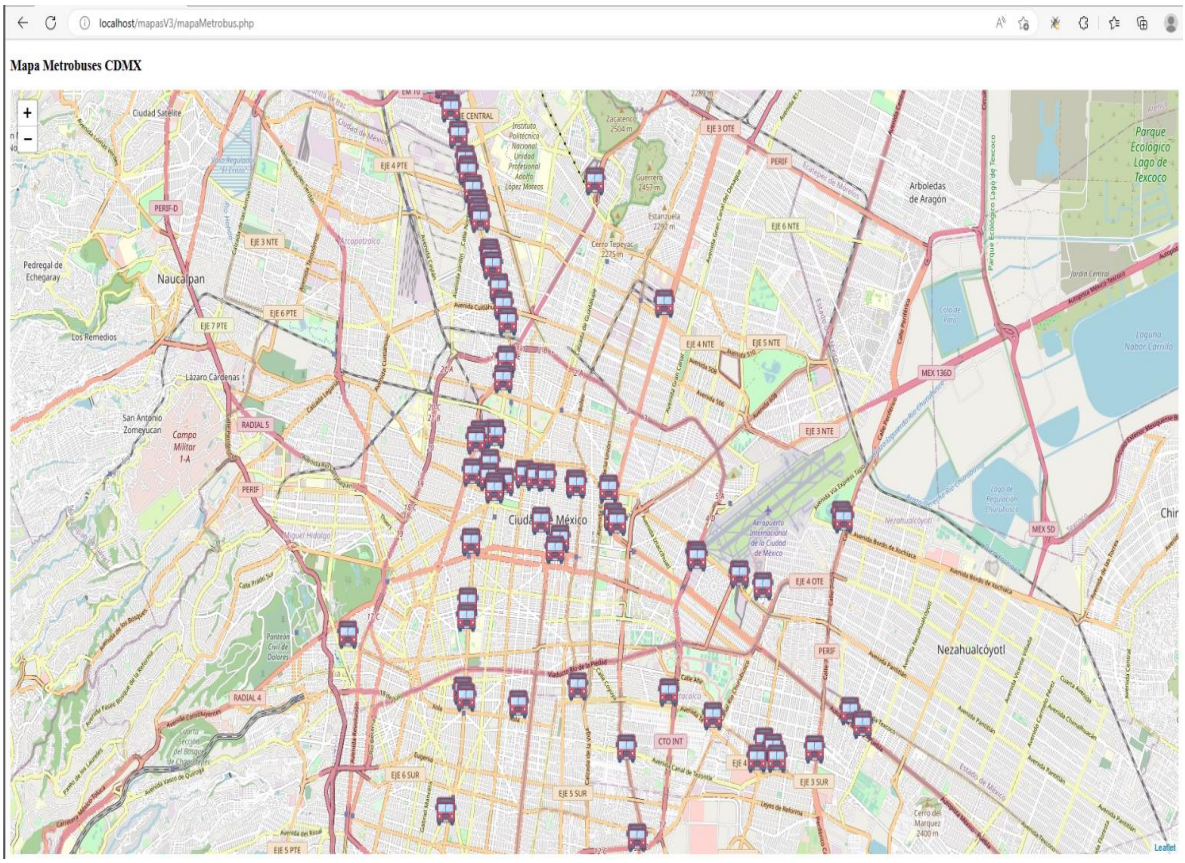


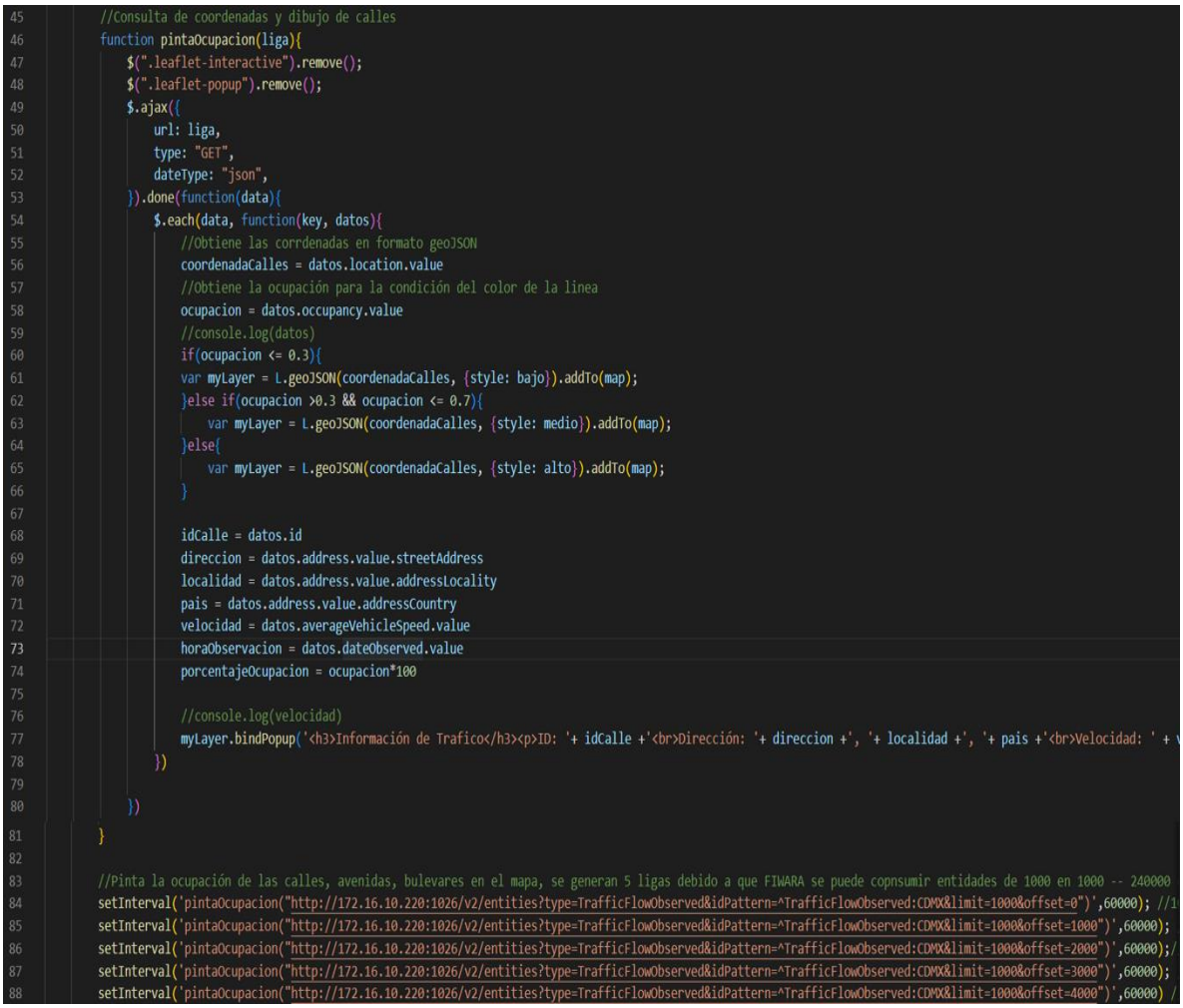
Figura 5.2 Representación del mapa con los Metrobuses

Mapa con datos de la congestión vehicular en tiempo real

Para la extracción de los datos en tiempo real de congestión vehicular de todos los segmentos de la Ciudad de México se desarrollaron las siguientes ligas.

1) http://ipServer:1026/v2/entities?type=*TrafficFlowObserved*&limit=1000&offset=0
2) http://ipServer:1026/v2/entities?type=*TrafficFlowObserved*&limit=1000&offset=1000
3) http://ipServer:1026/v2/entities?type=*TrafficFlowObserved*&limit=1000&offset=2000
4) http://ipServer:1026/v2/entities?type=*TrafficFlowObserved*&limit=1000&offset=3000
5) http://ipServer:1026/v2/entities?type=*TrafficFlowObserved*&limit=1000&offset=4000

Con esta liga se desarrolla una función *JavaScript* para poder obtener los datos en tiempo real y representarlos en un mapa. La Figura 5.3 es la función que desempeña esta actividad.



```
45      //Consulta de coordenadas y dibujo de calles
46      function pintaOcupacion(liga){
47          $(".leaflet-interactive").remove();
48          $(".leaflet-popup").remove();
49          $.ajax({
50              url: liga,
51              type: "GET",
52              dateType: "json",
53          }).done(function(data){
54              $.each(data, function(key, datos){
55                  //Obtiene las corrdenadas en formato geoJSON
56                  coordenadaCalles = datos.location.value
57                  //Obtiene la ocupación para la condición del color de la linea
58                  ocupacion = datos.occupancy.value
59                  //console.log(datos)
60                  if(ocupacion <= 0.3){
61                  var myLayer = L.geoJSON(coordenadaCalles, {style: bajo}).addTo(map);
62                  }else if(ocupacion >0.3 && ocupacion <= 0.7){
63                      var myLayer = L.geoJSON(coordenadaCalles, {style: medio}).addTo(map);
64                  }else{
65                      var myLayer = L.geoJSON(coordenadaCalles, {style: alto}).addTo(map);
66                  }
67
68                  idCalle = datos.id
69                  direccion = datos.address.value.streetAddress
70                  localidad = datos.address.value.addressLocality
71                  pais = datos.address.value.addressCountry
72                  velocidad = datos.averageVehicleSpeed.value
73                  horaObservacion = datos.dateObserved.value
74                  porcentajeOcupacion = ocupacion*100
75
76                  //console.log(velocidad)
77                  myLayer.bindPopup('<h3>Información de Trafico</h3><p>ID: '+ idCalle +'<br>Dirección: '+ direccion +', '+ localidad +', '+ pais +'<br>Velocidad: ' + v
78              })
79
80          })
81      }
82
83      //Pinta la ocupación de las calles, avenidas, bulevares en el mapa, se generan 5 ligas debido a que FIWARA se puede copnsumir entidades de 1000 en 1000 -- 240000
84      setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=0")',60000); //1
85      setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=1000")',60000);
86      setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=2000")',60000);/
87      setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=3000")',60000);
88      setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=4000")',60000) /
```

Figura 5.3 Función *JavaScript* que representa la congestión vehicular de todos los segmentos CDMX en un mapa

Las líneas 59 a la 66 se encargan de marcar los segmentos de la ciudad acorde a la ocupación de la calle. Cuando la ocupación es menor al 3% se pintará de color verde, cuando la ocupa es mayo al 3% y menor o igual al 7% se pintará de color amarillo y cuando la ocupación es mayor al 7% se pintará de color rojo.

Las líneas 68 a la 77 se encargan de extraer los datos de cada uno de los atributos de la entidad para poder representarlos en un cuadro de información.

La línea 83 a la 88 se encarga de invocar la función que pinta las entidades con las diferentes ligas que se crearon. Se realiza de esta forma ya que FIWARE solo permite adquirir datos de mil entidades a la vez. Esta se automatiza para que se ejecute cada 60 segundos.

En la Figura 5.4 se puede apreciar la representación de la congestión vehicular de toda la Ciudad de México.

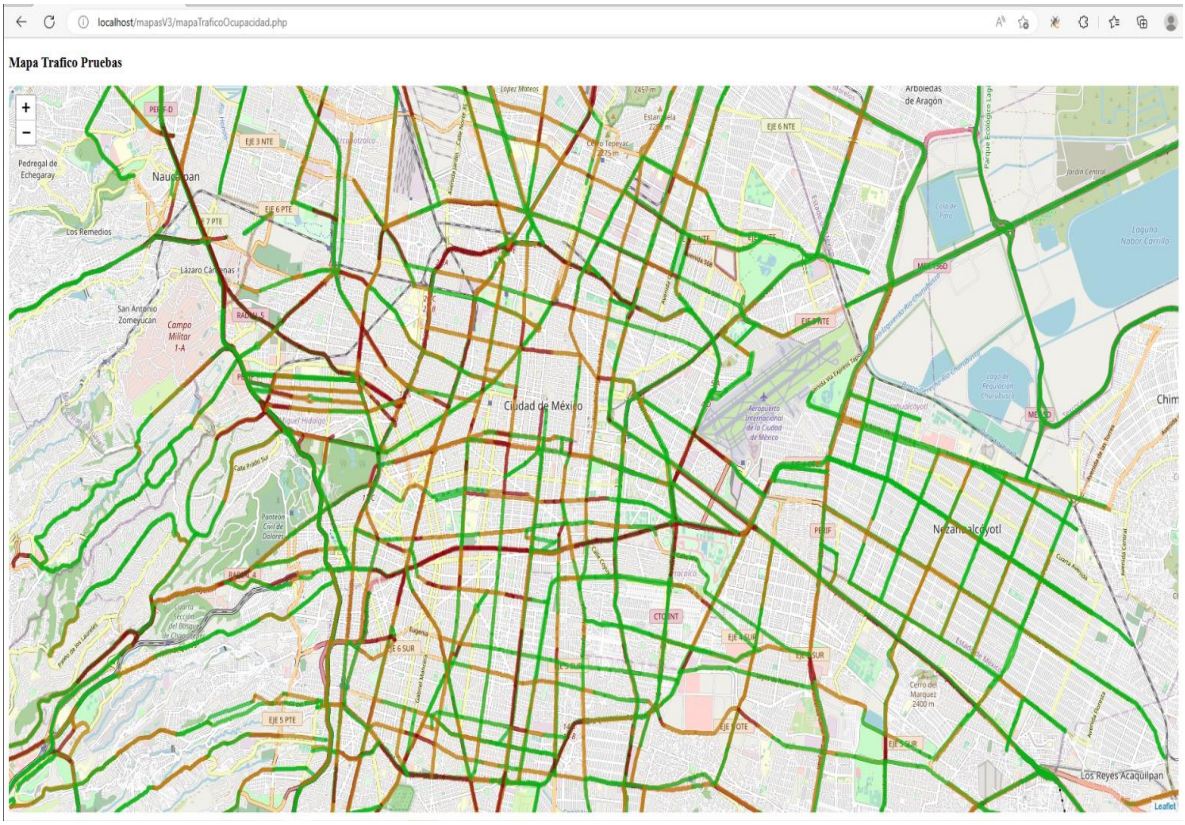


Figura 5.4 Representación del mapa con la congestión vehicular

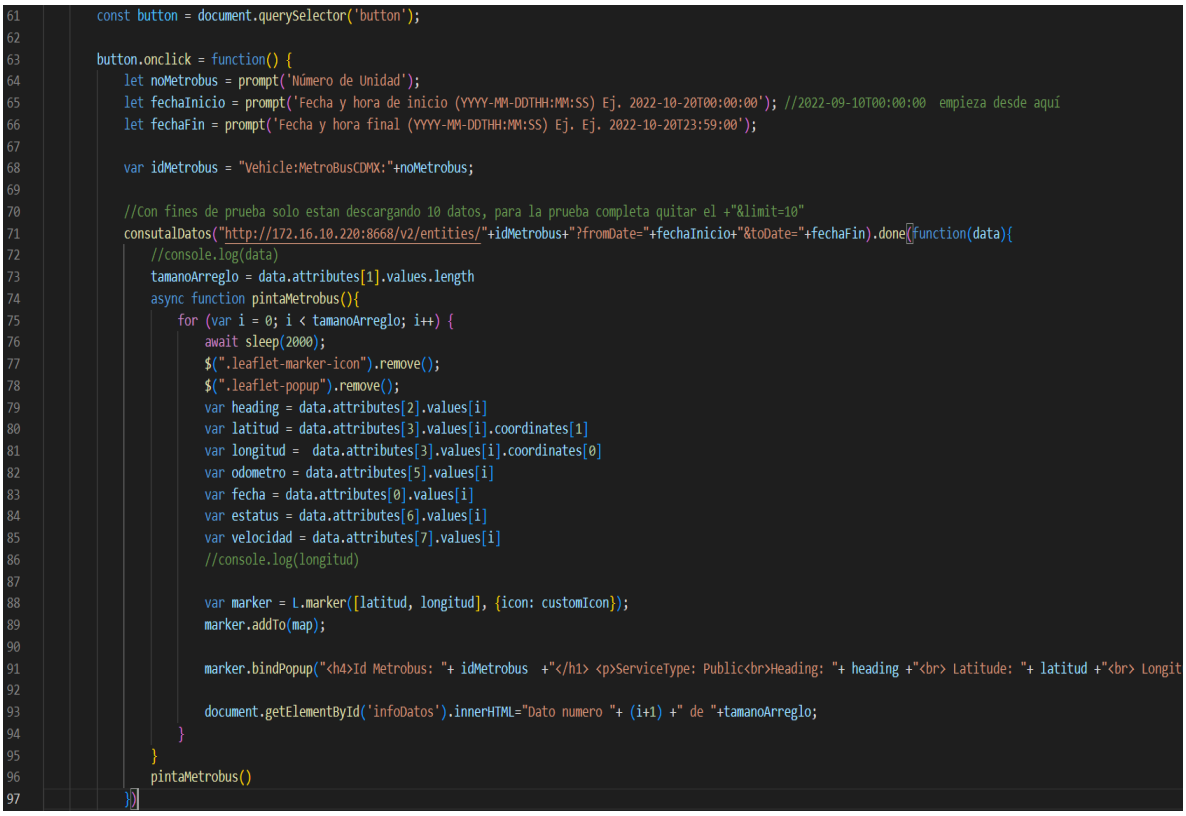
Mapa con datos históricos de un solo Metrobús

Para la extracción de los datos históricos de un solo Metrobús de la Ciudad de México se desarrolló la siguiente liga.

http://ipServer:8668/v2/entities/
Vehicle:MetrobusCDMX:**noMetrobus**?fromDate=**fechaInicio**&toDate=**fechaFin**

Esta liga cuenta con tres variables: (1) noMetrobus: representa el número de la unidad que se quiere monitorear. (2) fechaInicio: representa el punto de tiempo inicial del monitoreo. (3) fechaFin: representa el punto de tiempo final del monitoreo.

Con esta liga se desarrolló una función *JavaScript* para poder obtener los datos históricos y representarlos en un mapa. La Figura 5.5 es la función que desempeña esta actividad.



```
61      const button = document.querySelector('button');
62
63      button.onclick = function() {
64          let noMetrobus = prompt('Número de Unidad');
65          let fechaInicio = prompt('Fecha y hora de inicio (YYYY-MM-DDTHH:MM:SS) Ej. 2022-10-20T00:00:00'); //2022-09-10T00:00:00  empieza desde aquí
66          let fechaFin = prompt('Fecha y hora final (YYYY-MM-DDTHH:MM:SS) Ej. Ej. 2022-10-20T23:59:00');
67
68          var idMetrobus = "Vehicle:MetroBusCDMX:"+noMetrobus;
69
70          //Con fines de prueba solo estan descargando 10 datos, para la prueba completa quitar el +"&limit=10"
71          consutalDatos("http://172.16.10.220:8668/v2/entities/"+idMetrobus+"?fromDate="+fechaInicio+"&toDate="+fechaFin).done(function(data){
72              //console.log(data)
73              tamanoArreglo = data.attributes[1].values.length
74              async function pintaMetrobus(){
75                  for (var i = 0; i < tamanoArreglo; i++) {
76                      await sleep(2000);
77                      $(".leaflet-marker-icon").remove();
78                      $(".leaflet-popup").remove();
79                      var heading = data.attributes[2].values[i]
80                      var latitud = data.attributes[3].values[i].coordinates[1]
81                      var longitud =  data.attributes[3].values[i].coordinates[0]
82                      var odometro = data.attributes[5].values[i]
83                      var fecha = data.attributes[0].values[i]
84                      var estatus = data.attributes[6].values[i]
85                      var velocidad = data.attributes[7].values[i]
86                      //console.log(longitud)
87
88                      var marker = L.marker([latitud, longitud], {icon: customIcon});
89                      marker.addTo(map);
90
91                      marker.bindPopup("<h4>Id Metrobus: "+ idMetrobus  +"</h1> <p>ServiceType: Public<br>Heading: "+ heading +"<br> Latitude: "+ latitud +"<br> Longit
92
93                      document.getElementById('infoDatos').innerHTML="Dato numero "+ (i+1) +" de "+tamanoArreglo;
94                  }
95              }
96              pintaMetrobus()
97          })
```

Figura 5.5 *Script* para representar datos históricos de un Metrobús

Las líneas 61 a la 66 crean un botón que al oprimirlo abre una ventana donde solicita las tres variables mencionadas: número de unidad, fecha de inicio y fecha final. El formato de las fechas deben de ser YYYY-MM-DDTHH:mm:SS.

Las líneas 71 a la 96 se encargan de realizar la consulta de los datos históricos del número de unidad que se haya ingresado. Así mismo, las líneas 79 a la 85 se encargan de extraer los datos de cada una de las variables. La línea 91 agrega un cuadro con los datos obtenidos en la consulta.

En la Figura 5.6 se muestra los pasos de la ejecución del mapa que representa el comportamiento de un Metrobús en específico durante un tiempo dado. En total son cinco pasos.

Paso 1: Se pulsa el botón “Busca Metrobús”.

Paso 2: Se ingresa el número de la unidad que se quiere monitorear.

Paso 3: Se ingresa la fecha y hora de inicio del monitoreo.

Paso 4: Se ingresas la fecha y hora final del monitoreo.

Paso 5: Esperar un momento a que se realice la petición y en automático se ejecutara el recorrido realizado por el Metrobús elegido.

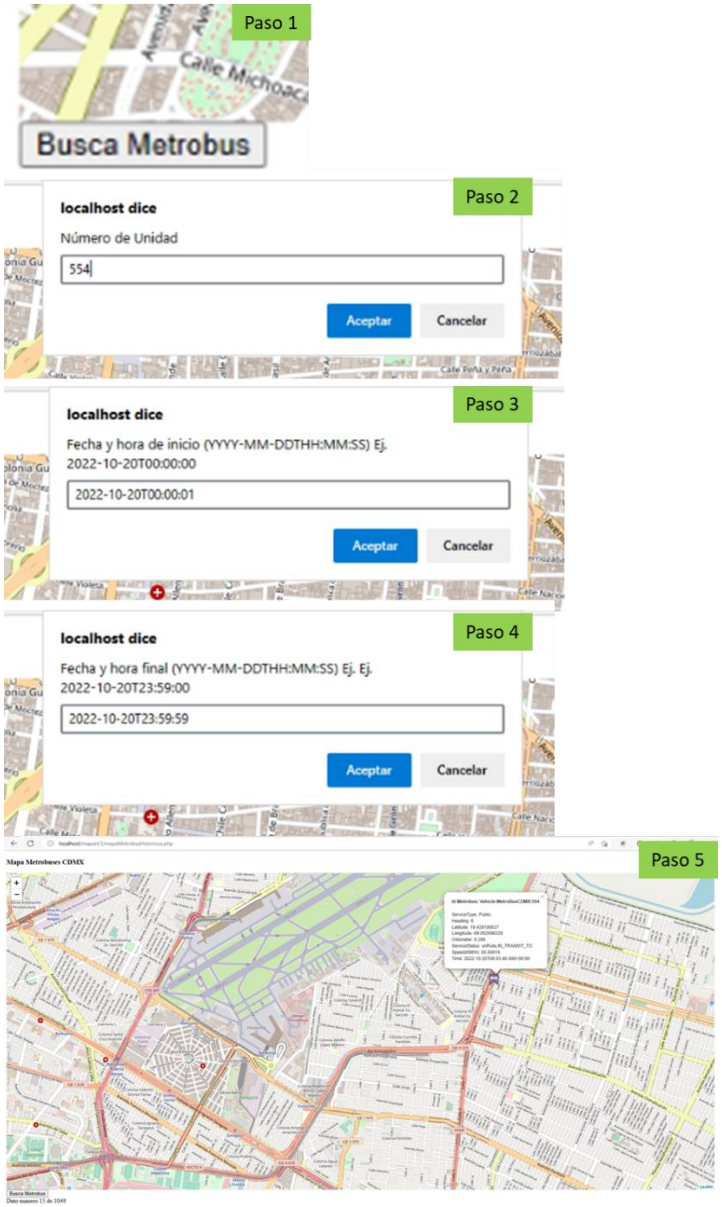


Figura 5.6 Ejecución del mapa con datos históricos de un Metrobús

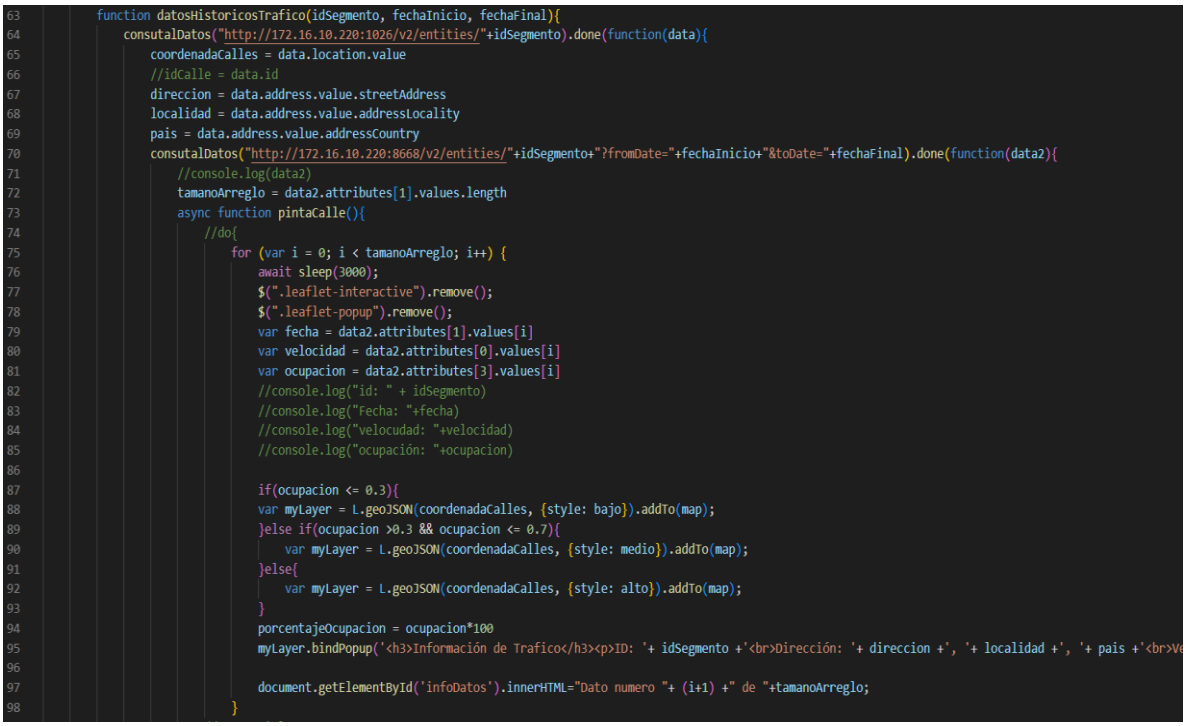
Mapa con datos históricos de congestión vehicular de un solo segmento de la ciudad

Para la extracción de los datos históricos de la congestión vehicular de un solo segmento de la Ciudad de México se desarrolla la siguiente liga.

http://ipServer:8668/v2/entities/**idSegmento**?fromDate=**fechaInicio**&toDate=**fechaFin**

Esta liga cuenta con tres variables: (1) idSegmento: representa el ID del segmento de la ciudad que se quiere monitorear. (2) fechaInicio: representa el punto de tiempo inicial del monitoreo. (3) fechaFin: representa el punto de tiempo final del monitoreo.

Con esta liga se desarrolla una función *JavaScript* para poder obtener los datos históricos y representarlos en un mapa. La Figura 5.7 es la función que desempeña esta actividad.



```
63      function datosHistoricosTrafico(idSegmento, fechaInicio, fechaFinal){
64          consutalDatos("http://172.16.10.220:1026/v2/entities/"+idSegmento).done(function(data){
65              coordenadaCalles = data.location.value
66              //idCalle = data.id
67              direccion = data.address.value.streetAddress
68              localidad = data.address.value.addressLocality
69              pais = data.address.value.addressCountry
70              consutalDatos("http://172.16.10.220:8668/v2/entities/"+idSegmento+"?fromDate="+fechaInicio+"&toDate="+fechaFinal).done(function(data2){
71                  //console.log(data2)
72                  tamanoArreglo = data2.attributes[1].values.length
73                  async function pintaCalle(){
74                      //do{
75                          for (var i = 0; i < tamanoArreglo; i++) {
76                              await sleep(3000);
77                              $(".leaflet-interactive").remove();
78                              $(".leaflet-popup").remove();
79                              var fecha = data2.attributes[1].values[i]
80                              var velocidad = data2.attributes[0].values[i]
81                              var ocupacion = data2.attributes[3].values[i]
82                              //console.log("id: " + idSegmento)
83                              //console.log("Fecha: "+fecha)
84                              //console.log("velocudad: "+velocidad)
85                              //console.log("ocupación: "+ocupacion)
86
87                              if(ocupacion <= 0.3){
88                              var myLayer = L.geoJSON(coordenadaCalles, {style: bajo}).addTo(map);
89                              }else if(ocupacion >0.3 && ocupacion <= 0.7){
90                                  var myLayer = L.geoJSON(coordenadaCalles, {style: medio}).addTo(map);
91                              }else{
92                                  var myLayer = L.geoJSON(coordenadaCalles, {style: alto}).addTo(map);
93                              }
94                              porcentajeOcupacion = ocupacion*100
95                              myLayer.bindPopup('<h3>Información de Trafico</h3><p>ID: '+ idSegmento +'<br>Dirección: '+ direccion +', '+ localidad +', '+ pais +'<br>Ve
96
97                              document.getElementById('infoDatos').innerHTML="Dato numero "+ (i+1) +" de "+tamanoArreglo;
98                          }
```

Figura 5.7 *Script* para representar datos históricos de la congestión vehicular de un segmento

Al igual que en el mapa de los históricos de los Metrobuses se crea un botón, el cual, al pulsarlo abre una venta donde solicita los valores: Id del segmento, fecha de inicio de monitoreo y fecha final del monitoreo.

Las líneas 64 a la 69 se encargan de extraer información común de todos los segmentos de la ciudad, esta información es la dirección, localidad, las coordenadas y el país.

Las líneas 70 a la 98 se encargan de extraer la información histórica del segmento que se eligió y lo representa en un mapa. Los colores de la representación son los mismos utilizados para la representación de los datos en tiempo real.

En la Figura 5.8**¡Error! No se encuentra el origen de la referencia.** se muestra los pasos de la ejecución del mapa que representa el comportamiento de un Metrobús en específico durante un tiempo dado. En total son cinco pasos.

Paso 1: Se pulsa el botón “Busca Segmento”.

Paso 2: Se ingresa el id del segmento de la Ciudad de México que se quiere monitorear.

Paso 3: Se ingresa la fecha y hora de inicio del monitoreo.

Paso 4: Se ingresas la fecha y hora final del monitoreo.

Paso 5: Esperar un momento a que se realice la petición y en automático se ejecutara el comportamiento de tráfico que tuvo el segmento durante el periodo de tiempo indicado.

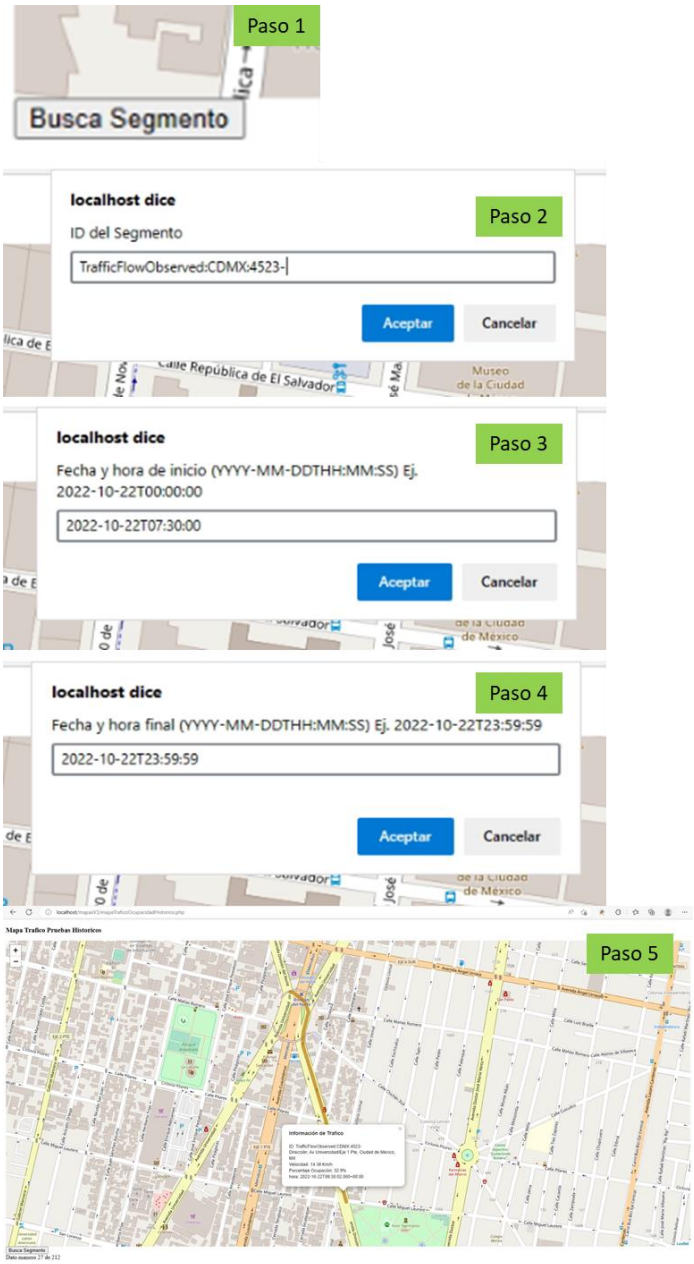


Figura 5.8 Ejecución del mapa con datos históricos de la congestión vehicular de un segmento

El código completo de cada uno de los mapas presentados se pueden observar en el anexo B denominado Alimentadores ADMU desarrollados.

5.2 Desarrollo de las consultas SQL para la monitorización de los datos históricos de forma gráfica

Para la conexión de Grafana hacia la base de datos de las entidades *Vehicle* se genera la conexión correspondiente entre Grafana y la base de datos donde se encuentran los datos históricos de las entidades *Vehicle*. En la Figura 5.9 se muestra la conexión dentro de Grafana. Cabe recordar que el tipo de base de datos que se utiliza para la conexión es *PostgraseSQL*.

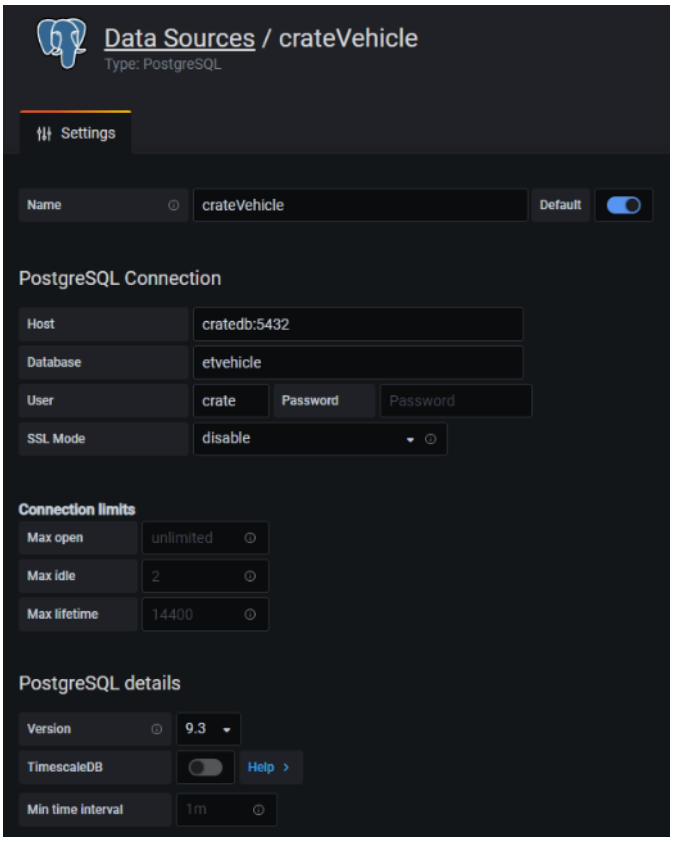


Figura 5.9 Conexión de Grafana a la base de datos histórica de las entidades *Vehicle*

Consulta SQL para monitoreo de velocidad del Metrobús

Ya que se cuenta con la conexión, se genera una consulta tipo SQL para la monitorización de la velocidad de algún Metrobús en específico. La consulta es:

SELECT "time_index" AS "time", "speed" AS "Metrobús: **noUnidad** " FROM "doc"."etvehicle" WHERE "entity_id" = 'Vehicle:MetrobusCDMX:**noUnidad**' AND $__timeFilter(time_index) ORDER BY 1 ASC;

En donde “noUnidad” es el número económico del Metrobús que se quiere monitorear. Con esta consulta se genera una gráfica serie tiempo donde se muestra como se ha comportado la velocidad del Metrobús durante un día específico. En la Figura 5.10 se puede apreciar el comportamiento de la velocidad del Metrobús 9406 durante el día 18/10/2022.

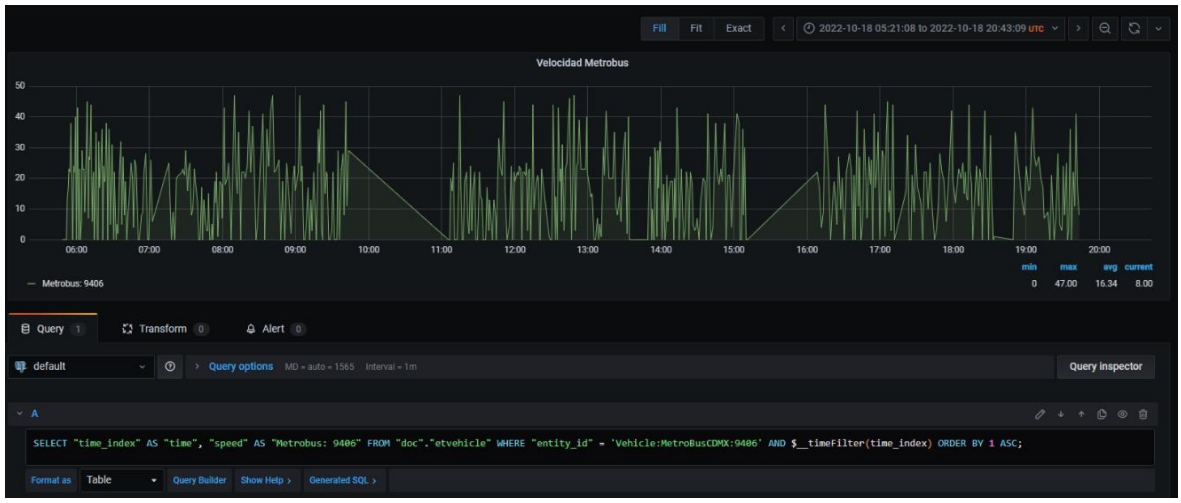


Figura 5.10 Gráfica serie tiempo del comportamiento de la velocidad del Metrobús 9406

Con esta consulta SQL se puede monitorear la velocidad de cualquier Metrobús de cualquier día. Solo basta con cambiar el valor “noUnidad” al número del Metrobús que se quiere monitorear. En la Figura 5.11 se muestra el comportamiento de la velocidad del Metrobús 2316 durante el día 09/11/2022.

Figura 5.11 Gráfica serie tiempo del comportamiento de la velocidad del Metrobús 2316

Conexión de Grafana hacia la base de datos de las entidades *TrafficFlowObserved*

Se genera la conexión correspondiente entre Grafana y la base de datos donde se encuentran los datos históricos de las entidades *TrafficFlowObserved*. En la Figura 5.12 se muestra la conexión dentro de Grafana. Cabe recordad que el tipo de base de datos que se utiliza para la conexión es *PostgraseSQL*.

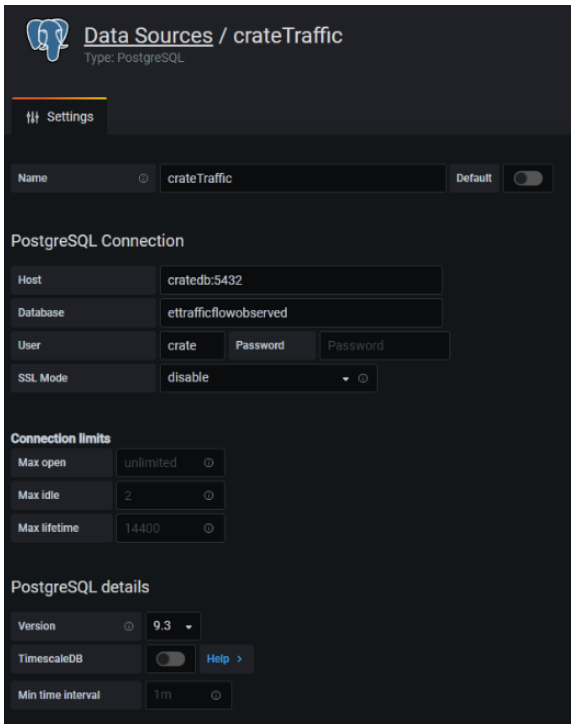


Figura 5.12 Conexión de Grafana a la base de datos histórica de las entidades *TrafficFlowObserved*

Consulta SQL para monitoreo de la velocidad del flujo de tráfico del segmento de la Ciudad

Una vez que se cuenta con la conexión hacia la base de datos de los datos históricos de las entidades *TrafficFlowObserved* se genera una consulta SQL con la que se puede monitorear la velocidad del flujo de tráfico de un segmento de la Ciudad de México. La consulta es:

SELECT "time_index" AS "time", "averagevehiclespeed" AS "*TrafficFlowObserved*:CDMX:**idSegmentoySentido**" FROM "doc"."et*TrafficFlowObserved*" WHERE "entity_id" = '*TrafficFlowObserved*:CDMX:**idSegmentoySentido**' AND $__timeFilter(time_index) ORDER BY 1 ASC;

En donde “idSegmentoySentido” es el id y el sentido del flujo del tráfico del segmento que se quiere monitorear. Con esta consulta se genera una gráfica serie tiempo donde se muestra como se ha comportado la velocidad del flujo de tráfico durante un día específico. En la Figura 5.13 se puede apreciar el comportamiento de la velocidad del flujo del tráfico del segmento 4386- durante el día 23/11/2022.

Figura 5.13 Gráfica serie tiempo del comportamiento de la velocidad del flujo de tráfico del segmento 4386-

Con esta consulta SQL se puede monitorear la velocidad del flujo de tráfico de cualquier segmento de la ciudad de cualquier día. Solo basta con cambiar el valor “idSegmentoySentido” al id y sentido del segmento de la ciudad que se quiere monitorear. En la Figura 5.14 se muestra el comportamiento de la velocidad del flujo del tráfico del segmento 14870+ durante el día 18/11/2022.

Figura 5.14 Gráfica serie tiempo del comportamiento de la velocidad del flujo de tráfico del segmento 14870+

Con estos ejercicios se valida que ambos alimentadores desarrollados funcionan de forma correcta, asimismo, se corrobora que en nuestro servidor FIWARE si está conservando datos en tiempo real en el *Orion Context Broker* y datos históricos en Quantumleap. Para mayores pruebas observar el Capítulo 6.

Capítulo 6

Pruebas y resultados del espacio unificado de datos

En este capítulo se analizan los datos históricos de movilidad urbana alojados en el espacio unificado para validar el funcionamiento correcto de los prototipos de software desarrollados y del espacio unificado.

6.1 Datos del espacio unificado

El espacio unificado de datos contiene datos históricos de la ubicación de los Metrobuses y la congestión vehicular de la Ciudad de México, de tal forma que se realizaron dos pruebas independientes: la evaluación del monitoreo de los datos de los Metrobuses y la evaluación del monitoreo de los datos de tráfico.

6.1.1 Datos de los Metrobuses de la Ciudad de México

Se realizó el monitoreo del total de los Metrobuses de la CDMX que cuentan con tecnología GPS. La cantidad de Metrobuses es variable a lo largo del día, pero se detectó que el número más grande de Metrobuses monitorizados en forma simultanea es de 823 unidades, sin embargo, el número promedio de Metrobuses que se monitorea cada día es de 750 unidades. Cada uno de las Metrobuses genera datos cada 65 segundo en promedio. Los Metrobuses está en operación por 18 horas en promedio por lo que se generan aproximadamente 900 lecturas al día de cada Metrobús. Cada Metrobús se identifica en forma independiente y se crea un histórico para cada uno de los Metrobuses.

6.1.2 Datos de la congestión vehicular de la Ciudad de México

Los datos de tráfico se obtienen como resultado de consultar la APIs de la aplicación HERE Maps. La Ciudad México se divide en un total de 4,133 segmentos que corresponden a calles y avenidas de la CDMX. No existe una correspondencia 1 a 1 entre un segmento y una calle, por ejemplo, una calle muy larga, como es el caso de Avenida de los Insurgentes se divide en múltiples segmentos. Existen calles pequeñas que si pueden ser especificadas con un solo segmento. Cada uno de los segmentos reporta el estado de la congestión vehicular en tiempo real de las calles. Los datos de estos segmentos se adquirieren cada 5 minutos y medio. Cada segmento de la ciudad genera su propio histórico de datos. Se obtienen datos 24 horas al día cada 5 minutos y medio, por lo que se generan un aproximado de 260 lecturas cada día por cada uno de los 4,133 segmentos de la CDMX.

6.2 Pruebas de continuidad de monitoreo

Se desarrollaron una serie de pruebas para medir la continuidad de monitoreo de las entidades alojadas en el espacio unificado de datos de movilidad. Con estas pruebas se midió el porcentaje del tiempo en que la fuente de datos produce información de forma correcta y el porcentaje de tiempo en la fuente de datos tiene problemas de continuidad en producir datos.

Es importante comentar que no se cuenta con datos faltantes en el espacio unificado de datos, es decir, cada valor que se produce de la fuente de datos es almacenada en el espacio unificado. Esto se comprobó comparando los registros de los archivos fuente contra los datos almacenados en el espacio unificado.

Como se ha comprobado, en el caso de los Metrobuses se presentan numerosos casos de falta de continuidad en los datos, es decir, espacios de tiempo donde un Metrobús deja de actualizar su información y como consecuencia se produce una discontinuidad en el tiempo en que ese Metrobús tuvo que haber producido datos.

6.2.1 Conjuntos de datos de prueba

Datos de los Metrobuses de la Ciudad de México

Los datos de los Metrobuses se descargan de la base de datos histórica de nuestro espacio unificado datos, los cuales corresponden a 30 Metrobuses monitorizados el día 20 de octubre del 2022.

Estos 30 Metrobuses han generado un conjunto de datos con 25,670 registros. Los 3 atributos que se tomaron en cuenta para las pruebas son los siguientes:

- time_index: indica la hora y fecha de la actualización de la entidad. Este atributo es de tipo datetime.
- speed: indica la velocidad en km/h que tiene la unidad al momento de la consulta. Este dato es de tipo numérico.
- id_entity: indica el id de la unidad. Este atributo es de tipo cadena.

Datos de congestión vehicular de la Ciudad de México

Para realizar las pruebas se descargaron de nuestro espacio unificado datos históricos del día 22 de octubre del 2022 correspondientes a 30 segmentos de la ciudad. De esta manera se obtiene un conjunto de datos de 7,469 registros. Los atributos utilizados para las pruebas de congestión vehicular son los siguientes:

- time_index: indica la hora y fecha de la actualización de la entidad. Este atributo es de tipo datetime.
- averagevehiclespeed: indica la velocidad en la fluye el trafico al momento de la consulta. Este atributo es de tipo numérico.
- id_entity: indica el id del segmento de la ciudad. Este atributo es de tipo cadena.

6.2.2 Fórmulas utilizadas para las pruebas de continuidad de monitoreo

Para evaluar la continuidad de los datos se utilizan fórmulas para: a) Transformación del formato de la fecha y hora a un número entero, b) Calcular el tiempo total monitoreado, c) Calcular la diferencia de tiempo que existe entre cada registro, d) Calcular el promedio de actualización de los datos, e) Calcular el tiempo no monitoreado, f) Calcular el porcentaje de tiempo monitoreado correctamente y porcentaje de tiempo no monitoreado.

a) Transformación del formato de la fecha y hora a un número entero

La fecha y hora de las actualizaciones son datos relevantes para nuestras pruebas, sin embargo, en nuestro espacio unificado los datos se encuentran almacenados en formato *datetime* por lo que se requirió realizar una trasformación al formato *Unix Timestamp*. Esta transformación nos permite

trabajar con la hora y fecha en un tipo de dato numérico. La transformación se logra utilizando librerías de la herramienta de análisis R. En la Tabla 6.1 se observa el ejemplo de la trasformación.

Tabla 6.1 Transformación de *datetime* a *Unix timestamp*

2022-10-20 05:04:32		1666260272

b) Calcular el periodo de tiempo que el Metrobús está en función

Para el caso de los Metrobuses se realiza el cálculo de la diferencia entre el último registro que se obtiene del monitoreo de un Metrobús y el primer registro en el cual el Metrobús comienza a producir datos. Esto con el fin de obtener el tiempo total, en segundos, en el que se realizó el monitoreo (fórmula 1).

$$tiempoTotalMonitoreado = time\_indes[tamañoArreglo] - time\_index[0] \quad (1$$

c) Calcular la diferencia de tiempo que existe entre cada registro

Se realiza este cálculo para obtener la diferencia de tiempo, en segundos, que se tiene entre cada actualización de datos. En el caso de los datos de Metrobuses se define que la actualización se realiza cada 30 segundos, sin embargo, se ha detectado que, en la práctica, este valor de la actualización de los datos de un Metrobús puede variar dependiendo la cantidad de Metrobuses que se encuentren generando datos de forma simultánea. En el caso del tráfico, se ha encontrado que la actualización el tiempo de actualización de los datos depende de la densidad de información de cada segmento de la ciudad, por ejemplo, los segmentos del centro de la ciudad contienen una densidad de información que los de una zona de la ciudad con menos afluencia vehicular (fórmula 2):

$$diferencia[i] = time\_index[i] - time\_index[i-1] \quad (2$$

Donde "i" es igual a la posición del arreglo de datos.

d) Calcular el promedio de actualización de los datos

Se calcula el valor promedio de las diferencias de tiempo entre registros para obtener el promedio de actualización, el resultado se obtiene en segundos (fórmula 3). Este dato es de utilidad para conocer la frecuencia con la que se han obtenido datos para un Metrobús específico o un segmento de la ciudad.

$$promedioActualizacion = \frac{\sum_{i=0}^{tamañoArreglo} diferencia[i]}{tamañoArreglo} \quad (3$$

Donde "i" es la posición del arreglo de datos.

e) Calcular el tiempo no monitoreado

Como se ha mencionado se corroboró no existen datos faltantes en nuestro registro, sin embargo, existen lapsos del tiempo donde los valores de un Metrobús o de un segmento de la ciudad no se han reportado a pesar de haberse cumplido el tiempo establecido para su actualización. En el caso de los Metrobuses esta situación se puede detectar al analizar que en dos o más valores de

actualización de un Metrobús no se ha modificado la hora de la actualización de datos, lo que indica que ese Metrobús no está generando datos en ese momento. En el caso de los datos de tráfico, se detectan datos no monitoreado en zonas donde existe mayor densidad de segmentos, esta densidad provoca que el prototipo desarrollado para tratar datos de congestión vehicular demore más tiempo en tratar los datos para enviarlos al servidor FIWARE.

Se calcula el tiempo no monitoreado a partir del promedio de actualización. Se toma en cuenta el promedio de actualización debido a ese dato representa el tiempo que una entidad tiene como margen para poder actualizarse (fórmula 4).

$$tiempoNoMonitoread = \sum_{i=0}^{tamañoArreglo} (diferencia[i] > promedioActualizacion) - promedioActualizacion \quad (4$$

Donde "i" es la posición del arreglo.

Con el objetivo de mejorar la comprensión del tiempo requerido para el tiempo monitoreado y no monitoreado se ha realizado su transformación de segundo a horas. Para el caso del tiempo de actualización se realizó la transformación de segundo a minutos.

f) Calcular el porcentaje de tiempo monitoreado correctamente y porcentaje de tiempo no monitoreado

Se calcula el porcentaje del tiempo que si se monitorea de forma correcta y el porcentaje donde se obtiene problemas de monitoreo (fórmula 5 y fórmula 6).

$$porcentajeTiempoNoReportado = \left(\frac{tiempoNoMonitoreado}{tiempoTotalMonitoreado}\right) \times 100 \quad (5$$

$$porcentajeTiempoSiReportado = 100 - porcentajeTiempoNoMonitoreado \quad (6$$

Cabe mencionar que estas fórmulas son ejecutadas en el conjunto de datos de los Metrobuses y de la congestión vehicular de la Ciudad de México.

6.2.3 Resultados obtenidos para el monitoreo de los Metrobuses de la CDMX

En la Tabla 6.2 se presenta un análisis estadístico sobre la velocidad de 30 Metrobuses. El objetivo es detectar si existen problemas de datos erróneos, como pueden ser velocidad negativas o nulas.

Tabla 6.2 Estadísticos de 30 Metrobuses de la CDMX

Identificador del Metrobús	Núm. Lecturas	Valores faltantes	Velocidad mínima (Km/h)	Velocidad máximo (Km/h)	Mediana de la velocidad (Km/h)	Media de la velocidad (Km/h)	Varianza de la velocidad	Desv. Estándar de la velocidad
1021	541.00	0	0.00	62.00	24.00	21.70	289.86	17.03
1302	422.00	0	0.00	64.00	17.00	17.77	235.96	15.36
607	1081.00	0	0.00	59.00	22.00	20.06	231.02	15.20
9402	571.00	0	0.00	47.00	20.00	17.93	146.62	12.11
807	1249.00	0	0.00	60.00	14.00	16.23	244.19	15.63
229	841.00	0	0.00	62.00	19.00	18.75	239.00	15.46
1112	424.00	0	0.00	63.00	22.00	20.71	309.38	17.59
628	976.00	0	0.00	57.00	19.00	19.48	203.45	14.26
774	982.00	0	0.00	64.00	19.00	18.50	230.13	15.17
1114	486.00	0	0.00	63.00	20.00	19.59	279.35	16.71
2607	445.00	0	0.00	56.00	24.00	24.30	166.42	12.90
791	1422.00	0	0.00	54.00	22.00	20.29	238.86	15.46
554	1049.00	0	0.00	74.00	10.00	14.95	286.83	16.94
602	833.00	0	0.00	59.00	24.00	23.14	251.65	15.86
822	1095.00	0	0.00	62.00	15.00	17.00	254.86	15.96
1310	485.00	0	0.00	67.00	15.00	16.66	248.39	15.76
2515	331.00	0	0.00	92.00	13.00	18.83	534.50	23.12
568	989.00	0	0.00	107.00	8.00	15.18	344.18	18.55
766	927.00	0	0.00	53.00	23.00	21.32	236.97	15.39
815	1222.00	0	0.00	54.00	15.00	15.59	213.46	14.61
904	876.00	0	0.00	53.00	13.00	15.15	221.27	14.88
768	1124.00	0	0.00	161.00	21.00	18.86	238.87	15.46
727	1350.00	0	0.00	69.00	21.00	20.56	307.56	17.54
470	462.00	0	0.00	49.00	15.50	15.63	188.02	13.71
398	886.00	0	0.00	66.00	19.00	18.98	273.58	16.54
3346	515.00	0	0.00	56.00	21.00	19.78	218.96	14.80
800	1238.00	0	0.00	55.00	23.00	21.10	222.66	14.92
827	1329.00	0	0.00	61.00	15.00	17.01	270.62	16.45
845	829.00	0	0.00	54.00	14.00	15.94	210.37	14.50
910	690.00	0	0.00	51.00	16.00	16.46	225.88	15.03

Se detectan 3 unidades que superaron la velocidad de 90 Km/h, lo que nos lleva a considerar que estos datos son considerados datos atípicos.

A continuación, se presenta el uso de las fórmulas utilizadas en las pruebas para describir los 30 autobuses utilizados como muestra en la experimentación. Cada tabla muestra al inicio el identificador del Metrobús analizado. Las gráficas presentadas permiten representar visualmente los segmentos donde no se presenta continuidad en la lectura de los datos. Como se ha mencionado anteriormente, la continuidad de los datos se ve afectada cuando el Metrobús no actualiza sus datos y, por lo tanto, existe una discontinuidad en el tiempo de monitoreo. Los casos más grandes de discontinuidad se presentan encerrado en círculos rojos, en algunos casos este tiempo sin actualización puede representar incluso varias horas, sin embargo, hay espacios de tiempo pequeños sin actualización que no es posible identificar a simple vista pero que si fueron considerados en nuestras fórmulas.

La parte izquierda de las gráficas representa el tiempo total monitoreado y la gráfica de la derecha representa una ampliación del 30% a los datos para presentar los primeros 300 registros del monitoreo de ese Metrobús. Esto se presenta para que pueda observarse que existen discontinuidades pequeñas que no son visibles en la gráfica general.

Es importante mencionar que se analizaron 30 casos para la prueba pero por simplicidad solo se muestran dos casos: uno del Metrobús con menos problemas de continuidad de datos (Tabla 6.3) y otro con el caso del Metrobús con más problemas de continuidad (Tabla 6.4). El total de las pruebas puede ser consultado en el Anexo C.

Tabla 6.3 Análisis de datos del Metrobús 791

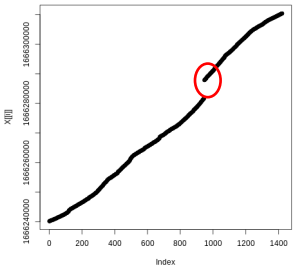
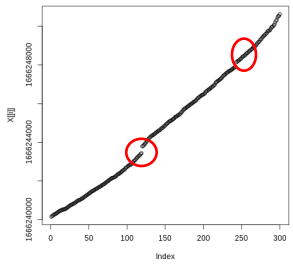
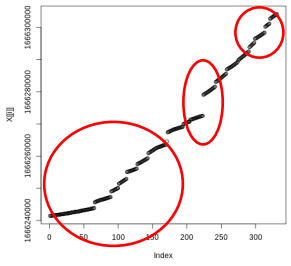
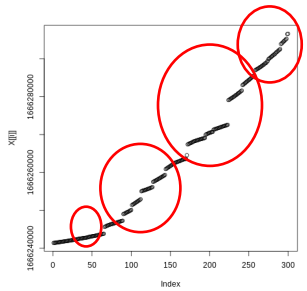
Vehicle:MetrobúsCDMX:791			
2022-10-20 04:29:11 Fecha y hora de inicio de monitoreo	2022-10-20 23:59:54 Fecha y hora final del monitoreo	70243 Periodo de funcionamiento del Metrobús (segundos)	18790.09 Total de tiempo no reportado (segundos)
49.4 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
19.51 Periodo de funcionamiento del Metrobús (horas)	0.82 Tiempo promedio de actualización (minutos)	5.22 Tiempo no reportado (horas)	27 Porcentaje de tiempo no reportado

Tabla 6.4 Análisis de datos del Metrobús 2515

Vehicle:MetrobúsCDMX:2515			
2022-10-20 04:50:21 Fecha y hora de inicio de monitoreo	2022-10-20 22:16:01 Fecha y hora final del monitoreo	62740 Periodo de funcionamiento del Metrobús (segundos)	27699.05 Total de tiempo no reportado (segundos)
189.55 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
17.43 Periodo de funcionamiento del Metrobús (horas)	3.16 Tiempo promedio de actualización (minutos)	7.69 Tiempo no reportado (horas)	44 Porcentaje de tiempo no reportado

La Tabla 6.5 muestra el porcentaje del tiempo no monitoreado y el porcentaje del tiempo monitoreado de los 30 Metrobuses de la CDMX.

Tabla 6.5 Resumen del análisis de datos de 30 Metrobuses de la CDMX

ID unidad	Periodo de funcionamiento del Metrobús		Total de tiempo no reportado		Tiempo promedio de actualización		Porcentaje tiempo no reportado	Porcentaje de tiempo reportado
	Segundos	Horas	Segundos	Horas	Segundos	Minutos	%	%
Metrobús 1021	66564	18.49	16035	4.45	123	2.05	24	76
Metrobús 1302	54506	15.14	9510	2.64	129	2.15	17	83
Metrobús 607	63934	17.76	20540	5.71	59	0.98	32	68
Metrobús 9402	66108	18.36	14641	4.07	116	1.93	22	78
Metrobús 807	64357	17.88	16381	4.55	52	0.87	25	75
Metrobús 229	58536	16.26	19613.13	5.45	69.6	1.16	34	66
Metrobús 1112	61756	17.15	16535.28	4.59	145.65	2.43	27	73
Metrobús 628	64696	17.97	22410.89	6.23	66.29	1.10	35	65
Metrobús 774	59261	16.46	16409.85	4.56	60.35	1.01	28	72
Metrobús 1114	65316	18.14	15696.1	4.36	134.4	2.24	24	76
Metrobús 2607	54599	15.17	11994.01	3.33	122.69	2.04	22	78
Metrobús 791	70243	19.51	18790.09	5.22	49.4	0.82	27	73
Metrobús 554	62113	17.25	17180.19	4.77	59.21	0.99	28	72
Metrobús 602	62671	17.41	27277.82	7.58	75.24	1.25	44	56
Metrobús 822	60704	16.86	14104.59	3.92	55.44	0.92	23	77
Metrobús 1310	59562	16.55	10405.19	2.89	122.81	2.05	17	83
Metrobús 2515	62740	17.43	27699.05	7.69	189.55	3.16	44	56
Metrobús 568	61321	17.03	17263.38	4.80	62	1.03	28	72
Metrobús 766	57141	15.87	18455.61	5.13	61.64	1.03	32	68
Metrobús 815	66756	18.54	16782.39	4.66	54.63	0.91	25	75
Metrobús 904	67964	18.88	22116.06	6.14	77.58	1.29	33	67
Metrobús 768	60164	16.71	14879.16	4.13	53.53	0.89	25	75
Metrobús 727	65808	18.28	15597.99	4.33	48.75	0.81	24	76
Metrobús:470	56648	15.74	33338.24	9.26	122.61	2.04	59	41
Metrobús 398	61538	17.09	20262.01	5.63	69.46	1.16	33	67
Metrobús 3346	63598	17.67	13552.24	3.76	123.49	2.06	21	79
Metrobús 800	65941	18.32	13572.74	3.77	53.26	0.89	21	79
Metrobús 827	67446	18.74	15477	4.30	50.75	0.85	23	77
Metrobús 845	61236	17.01	18353	5.10	73.87	1.23	30	70
Metrobús 910	61042	16.96	18477	5.13	88.47	1.47	30	70

Finalmente, la Tabla 6.6 muestra el porcentaje del tiempo promedio en donde se monitorea de forma correcta y en donde no se monitorea de forma correcta.

Tabla 6.6 Promedio del porcentaje de tiempo de monitoreo para los Metrobuses

Promedio total del funcionamiento del Metrobús (horas)	Promedio total de tiempo no monitoreado (horas)	Tiempo promedio de actualización (minutos)	Porcentaje promedio tiempo no monitoreado	Porcentaje promedio de tiempo monitoreado
17.35	4.94	1.43	28.57	71.43

6.2.4 Resultados obtenidos para el monitoreo de la congestión vehicular de la CDMX

En la Tabla 6.7 se presenta un análisis estadístico sobre la velocidad de trafico de 30 segmentos de la Ciudad de México. El objetivo es detectar si existen problemas de datos erróneos, como pueden ser velocidades negativas o nulas.

Tabla 6.7 Estadísticos de 30 segmentos de la CDMX

ID segmento	Núm. Lecturas	Valores faltantes	Velocidad mínima (Km/h)	Velocidad máximo (Km/h)	Mediana de la velocidad (Km/h)	Media de la velocidad (Km/h)	Varianza de la velocidad	Desv. Estándar de la velocidad
4494-	246.00	0	8.93	31.07	17.52	19.24	29.32	5.42
11392-	108.00	0	40.33	45.61	45.61	45.40	0.63	0.79
12952-	348.00	0	11.82	31.07	29.46	28.48	11.31	3.36
4505-	108.00	0	4.39	49.71	39.74	31.48	323.00	17.97
14519+	108.00	0	21.68	49.71	43.17	42.57	31.37	5.60
13125-	353.00	0	2.42	31.07	14.21	14.10	54.06	7.35
4500-	332.00	0	30.00	49.71	47.27	46.13	16.75	4.09
4523-	332.00	0	5.50	31.07	17.77	18.70	34.40	5.87
13213-	353.00	0	3.71	31.07	14.32	15.15	71.41	8.45
4495-	246.00	0	2.42	31.07	16.44	17.57	37.78	6.15
4296-	332.00	0	9.79	28.44	16.80	17.61	19.45	4.41
13110-	353.00	0	8.52	31.07	22.14	21.27	35.95	6.00
4107-	246.00	0	4.03	45.34	25.24	25.00	134.34	11.59
13437-	246.00	0	3.53	43.99	10.54	16.35	126.82	11.26
4363-	332.00	0	9.80	29.35	19.37	20.02	16.56	4.07
4284+	108.00	0	5.35	27.61	14.83	15.58	22.46	4.74
13034-	353.00	0	9.39	31.07	19.31	20.40	29.18	5.40
13169-	353.00	0	2.42	31.07	9.81	11.89	50.37	7.10
4474+	108.00	0	6.11	49.71	34.39	31.82	181.04	13.45
5147-	108.00	0	2.42	31.07	14.61	15.56	78.06	8.84
4366+	332.00	0	5.84	31.07	14.28	15.44	15.44	5.10
13176-	353.00	0	11.19	31.07	19.80	21.58	24.27	4.93
4485+	108.00	0	10.15	49.71	23.80	30.65	30.65	13.88
4895-	332.00	0	10.49	26.41	15.41	16.20	15.16	3.89
4510-	108.00	0	2.42	31.07	17.14	17.30	69.42	8.33
12969+	108.00	0	12.43	31.07	22.81	22.79	35.85	5.99
4237+	108.00	0	8.11	49.71	28.37	30.79	229.13	15.14
13165-	353.00	0	7.01	31.07	21.79	20.86	51.01	7.14
12947-	348.00	0	2.60	31.07	15.77	17.23	79.66	8.93
4219+	246.00	0	7.04	30.83	16.70	17.82	36.36	6.03

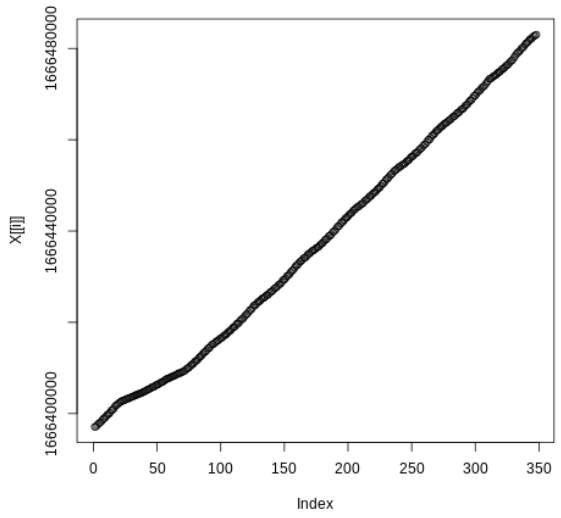
De esta manera se descarta la posibilidad de que existan datos faltantes y velocidades fuera de rango. De igual forma se puede observar el número de registros que se tienen por cada segmento, así como su velocidad máxima, velocidad mínima y su velocidad promedio.

Es importante mencionar que se analizaron 30 casos para la prueba pero por simplicidad solo se muestran dos casos: uno donde el segmento analizado se encuentra en una zona con poca densidad de segmentos (Tabla 6.8) y otro donde el segmento analizado se encuentra en una zona con mucha densidad de segmentos (Tabla 6.9). El total de las pruebas puede ser consultado en el Anexo C.

Cada tabla muestra al inicio el identificador del segmento el cual se está analizando. La gráfica presentada permite representar visualmente los segmentos donde no se presenta continuidad en la lectura de los datos. Como se mencionó anteriormente, la continuidad de los datos se ve afectada cuando existe gran densidad de segmentos en una zona, por lo tanto, existe una discontinuidad en el tiempo de monitoreo. Existen espacios de tiempo pequeños sin actualización que no es posible identificar a simple vista pero que si fueron considerados en nuestras fórmulas.

Tabla 6.8 Análisis de datos del segmento 12952-

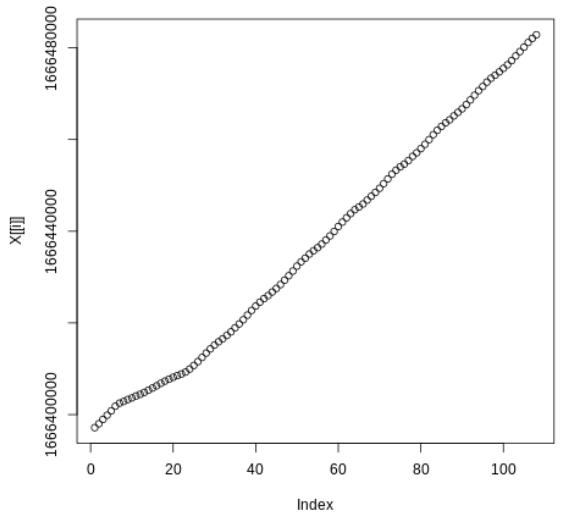
TrafficFlowObserved:CDMX:12952-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	11701 Total de tiempo no monitoreado (segundos)
246.89 Tiempo promedio de actualización (segundos)			



Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.

23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	3.25 Tiempo no monitoreado (horas)	10.39 Porcentaje de tiempo no monitoreado

Tabla 6.9 Análisis de datos del segmento 4505-

TrafficFlowObserved:CDMX:4505-			
2022-10-22 00:06:02 Fecha y hora de inicio de monitoreo	2022-10-22 23:54:02 Fecha y hora final del monitoreo	85680 Total de tiempo monitoreado (segundos)	8747 Total de tiempo no monitoreado (segundos)
793.33 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.8 Tiempo de monitoreo (horas)	13.22 Tiempo promedio de actualización (minutos)	2.43 Tiempo no monitoreado (horas)	10.21 Porcentaje de tiempo no monitoreado

A continuación, se presenta una tabla resumida que muestra el porcentaje del tiempo no monitoreado y el porcentaje del tiempo monitoreado de los 30 segmentos de la CDMX. Observar Tabla 6.10.

Tabla 6.10 Resumen del análisis de datos de 30 segmentos de la CDMX

ID segmento	Total de tiempo monitoreado		Total de tiempo no monitoreado		Tiempo promedio de actualización		Porcentaje tiempo no monitoreado	Porcentaje de tiempo monitoreado
	Segundos	Horas	Segundos	Horas	Segundos	Minutos	%	%
Segmento 4494-	85919	23.87	8930	2.48	349.26	5.82	10.39	89.61
Segmento 11392-	85680	23.80	8747	2.43	793.33	13.22	10.21	89.79
Segmento 12952-	85919	23.87	11701	3.25	246.89	4.11	10.39	89.61
Segmento 4505-	85680	23.80	8747	2.43	793.33	13.22	10.21	89.79
Segmento 14519+	85680	23.80	8747	2.43	793.33	13.22	10.21	89.79
Segmento 13125-	85919	23.87	8248	2.29	243.4	5.50	9.6	90.40
Segmento 4500-	85919	23.87	8248	2.29	258.79	5.50	9.6	90.40
Segmento 4523-	85919	23.87	8248	2.29	258.79	5.50	9.6	90.40
Segmento 13213-	85919	23.87	8248	2.29	243.4	5.50	9.6	90.40
Segmento 4495-	85919	23.87	8248	2.29	349.26	5.82	9.6	90.40
Segmento 4296-	85919	23.87	8248	2.29	258.79	5.50	9.6	90.40
Segmento 13110-	85919	23.87	8248	2.29	243.4	4.06	9.6	90.40
Segmento 4107-	85919	23.87	8930	2.48	349.26	5.82	10.39	89.61
Segmento 13437-	85919	23.87	8930	2.48	349.26	5.82	10.39	89.61
Segmento 4363-	85919	23.87	8248	2.29	258.79	5.50	9.6	90.40
Segmento 4284+	85680	23.80	8747	2.43	793.33	13.22	10.21	89.79
Segmento 13034-	85919	23.87	8248	2.29	243.4	5.50	9.6	90.40
Segmento 13169-	85919	23.87	8248	2.29	243.4	5.50	9.6	90.40
Segmento 4474+	85680	23.80	8747	2.43	793.33	13.22	10.21	89.79
Segmento 5147-	85680	23.80	8747	2.43	793.33	13.22	10.21	89.79
Segmento 4366+	85919	23.87	8248	2.29	258.79	5.50	9.6	90.40
Segmento 13176-	85919	23.87	8248	2.29	243.4	5.50	9.6	90.40
Segmento 4485+	85680	23.80	8747	2.43	793.33	13.22	10.21	89.79
Segmento 4895-	85919	23.87	8248	2.29	258.79	5.50	9.6	90.40
Segmento 4510-	85680	23.80	8747	2.43	793.33	13.22	10.21	89.79
Segmento 12969+	85680	23.80	8747	2.43	793.33	13.22	10.21	89.79
Segmento 4237+	85680	23.80	8747	2.43	793.33	13.22	10.21	89.79
Segmento 13165-	85919	23.87	8248	2.29	243.4	5.50	9.6	90.40
Segmento 12947-	85919	23.87	8248	2.29	246.89	5.50	9.6	90.40
Segmento 4219+	85919	23.87	8930	2.48	349.26	5.82	10.39	89.61

Finalmente, la Tabla 6.11 muestra el porcentaje del tiempo promedio en donde se monitorea la información de forma correcta y en donde no se monitorea de forma correcta.

Tabla 6.11 Promedios del porcentaje de tiempo monitoreado para los segmentos de la CDMX

Promedio total de tiempo monitoreado (horas)	Promedio total de tiempo no monitoreado (horas)	Tiempo promedio de actualización (minutos)	Porcentaje promedio tiempo no monitoreado	Porcentaje promedio de tiempo monitoreado
23.84	2.39	8.03	9.94	90.07

Capítulo 7

Conclusiones y trabajo futuro

En esta sección se presentan las conclusiones que se han generado a partir de este proyecto de tesis. Se describen también los trabajos futuros que se derivan de la investigación realizada en la tesis.

7.1 Conclusiones

Uno de los problemas de los proyectos de Ciudades Inteligentes es que no existen estándares para la representación de la información producida por los múltiples sensores que toman datos de contexto. La implementación de soluciones IoT sin estándares incrementa los costos del proyecto, pone en riesgo la adopción de soluciones a gran escala y limita de manera importante la innovación tecnológica de iniciativas de Ciudades Inteligentes.

Algunas ciudades hacen mención de haber desarrollado iniciativas de Ciudades Inteligentes en los últimos años, pero en pocos casos encontramos una implementación de estándares para la gestión de los datos. Esta ausencia de estándares hace que una solución que funciona en una ciudad, no pueda funcionar en otra sin esfuerzos de adaptación muy relevantes.

El espacio unificado de datos propuesto utiliza tecnologías de la solución FIWARE. Las soluciones basadas en FIWARE buscan impulsar estándares para recopilar, gestionar y publicar información de contexto y adicionalmente aportar elementos que permiten explotar esta información una vez recopilada. El uso de estándares FIWARE puede resultar clave para construir un mercado digital único para las aplicaciones inteligentes donde las apps y soluciones puedan adaptarse de un cliente a otro sin grandes cambios.

El desarrollo de estándares no existe en la actualidad yes resulta clave para construir un mercado digital único para las aplicaciones inteligentes donde las aplicaciones y soluciones puedan portarse de un cliente a otro sin grandes cambios.

En este sentido, el espacio unificado de datos y los algoritmos desarrollados en este proyecto de tesis contribuyen a la resolución de la complejidad de tratar información recogida por sensores y traducirlos a un lenguaje común, estandarizando la representación y la publicación de los datos.

Adicionalmente, el espacio unificado es capaz de exportar y publicar parte de la información de movilidad que contiene para que terceros puedan desarrollar aplicaciones interesantes y de utilidad para el ciudadano, para la economía local y para los procesos productivos de la movilidad de la Ciudad de México.

El espacio unificado de datos desarrollado en este proyecto de tesis contiene información estandarizada de datos de la movilidad de la Ciudad de México que pueden consultarse mediante servicios *REST*, logrando de esta manera el objetivo planteado en el proyecto de tesis. Sin embargo, algunas de las características de los datos y sus fuentes de origen provocan que la adquisición, mapeo y envío de datos a FIWARE tengan un comportamiento inusual. Algunos de los puntos encontrados son:

Tiempos altos para la extracción de los datos

Este proyecto de tesis es dependiente de las fuentes de datos de movilidad. Se esperaba que la adquisición de los datos no superara más de 10 segundos, sin embargo, durante su monitoreo se

detecta que para la descarga de los datos de los Metrobuses en ocasiones se consume aproximadamente 60 segundos. Este comportamiento se contribuye a la infraestructura de red del origen.

Entidades con información no constante

Lo que se esperaba de los datos de los Metrobuses de la Ciudad de México es que compartieran su información de forma constante y de esta forma tener un tiempo de actualización promedio más cercanos a los 65 segundos. Sin embargo, la falta de esta constancia de algunos Metrobuses provoca que el promedio de actualización aumente, ya que, por lapsos de tiempo aleatorios, algunas unidades dejan de compartir su información.

Gran número de entidades

Lo que se esperaba es que el prototipo desarrollado para mapear datos de la congestión vehicular de la Ciudad de México tuviera la capacidad suficiente para transformar las más de 4,000 segmentos al modelo de datos *TrafficFlowObserved* en un tiempo aproximado de 5 minutos. Sin embargo, la cantidad de datos que se deben de consumir, mapear y enviar a la Plataforma FIWARE es considerablemente grande. Esta gran cantidad de datos provoca un gran consumo de recursos computacionales.

Pese a estas complicaciones en las fuentes de datos y su respectivo procesamiento, se logró poner a disposición datos en tiempo real y datos históricos de la ubicación de los Metrobuses y la congestión vehicular de la Ciudad de México. El acceso a los datos estandarizados de la ubicación y congestión vehicular de la Ciudad de México que se encuentran alojados en nuestro espacio unificado es sencillo a comparación de las dos fuentes de datos.

El acceso a estos datos es relativamente sencillo a comparación de los orígenes de datos.

De igual manera, el software desarrollado puede adaptarse (con pocas modificaciones) para poder tratar datos GTFS-RT y datos de congestión vehicular proveniente de HEREMaps de cualquier zona geográfica que tenga alcance a estas tecnologías.

Los métodos desarrollados para la representación de los datos de forma gráfica facilitan el análisis, comprensión y detección de inconvenientes relacionadas a la movilidad urbana. De esta manera ayudan a la toma de decisiones para acciones futuras que ayuden a la mejora de la movilidad urbana.

7.2 Trabajo futuro

Como se mencionó, este proyecto de tesis contribuye a la resolución de la complejidad de tratar información de los diferentes dispositivos IoT y traducirlos en un lenguaje común. Específicamente, nuestro proyecto trabaja con dos fuentes de datos: (1) Datos de la ubicación de los Metrobuses de la Ciudad de México. (2) Datos de congestión vehicular de la Ciudad de México.

Como estas fuentes existen muchas más, las cuales comparten información y la ponen a disposición de la sociedad, sin embargo, estos datos no cuentan con estándares que faciliten su reutilización. Es por ello que deben de existir más proyectos que implementen la estandarización de los datos IoT.

La estandarización de los datos de movilidad permitirá la implementación de soluciones inteligentes para la construcción de ciudades inteligentes que puedan solventar problemas de la vida real.

Adicionalmente, los prototipos desarrollados en este proyecto de tesis pueden ser optimizados para poder extraer, mapear y enviar los datos a FIWARE y de esta manera poder trabajar con los datos más cercanos al tiempo real.

Asimismo, los datos de movilidad almacenados en nuestro espacio unificado se encuentran disponibles para un análisis profundo o inclusive la generación de alguna aplicación que genere un bien que beneficie de la movilidad de la sociedad de la Ciudad de México.

7.3 Logros

- Aceptación y presentación del artículo titulado: *"Servicio para consumir datos de transporte público en tiempo real utilizando la plataforma FIWARE"*. En el XIV CONGRESO MEXICANO DE INTELIGENCIA ARTIFICIAL realizado del 25 al 27 de mayo de 2022, de forma virtual.
- Publicación del artículo: Luis Reyes-González, Hugo Estrada-Esquivel, Alicia Martinez-Rebollar y Yasmín Hernández-Pérez, 2022. Servicio para consumir datos de transporte público en tiempo real utilizando la Plataforma FIWARE. Research in Computing Science, Vol. 151(5), pp. 159-172. ISSN: 1870-4069. Research in Computing Science 151(5) (ipn.mx)

Referencias

1. REDEUSLAC. (2021, 20 junio). Economía Urbana. https://redeuslac.org/lineas-de-accion/economia-urbana/#:%7E:text=Las%20ciudades%2C%20que%20concentran%20capital,productividad%20y%20la%20capacidad%20innovadora. Consultado: marzo 2022.
2. Montezuma, R. (2003). Ciudad y transporte: la movilidad urbana. https://repositorio.cepal.org/handle/11362/27823 Consultado: marzo 2022.
3. Colchado Flores, I. C. F. (2017, 28 abril). La movilidad urbana en la Ciudad de México: un problema complejo. Centro de Ciencias de la Complejidad. https://www.c3.unam.mx/boletines/boletin5.html#:%7E:text=Las%20causas%20detr%C3%A1s%20del%20problema,comportamiento%20de%20los%20usuarios%20y Consultado marzo 2022.
4. Pública, A. D. D. I. (2021). Portal de Datos Abiertos de la CDMX. Portal de Datos Abiertos de la CDMX. https://datos.cdmx.gob.mx/ Consultado: abril 2022.
5. Google. (2021, 10 febrero). Referencia de GTFS Real-time | Transporte en tiempo real |. Google Developers. https://developers.google.com/transit/gtfs-realtime/reference?hl=es-419 Consultado abril 2022.
6. FIWARE. (2020). Modelos de datos - Aprende FIWARE en español. Aprendiendo FIWARE En español. https://fiware-training.readthedocs.io/es_MX/latest/ecosistemaFIWARE/modelosdedatos/#:%7E:text=Los%20modelos%20de%20datos%20se,con%20FIWARE%20NGSI%20versi%C3%B3n%202. Consultado abril 2022.
7. HERE Maps. (2022). Documentation, Code Examples and API References. HERE Develo-per. https://developer.here.com/documentation?cid=www.here.com-topnav-menu-docu-mentation Consultado: mayo 2022.
8. Metrobuses CDMX. (2022). Datos Abiertos. Metrobús. https://www.metrobus.cdmx.gob.mx/portal-ciudadano/datos-abiertos Consultado agosto 2022.
9. KaaIoT. (2022). Enterprise IoT Platform with Free Plan | Kaa. Kaa IoT Platform. https://www.kaaiot.com/ Consultado: mayo 2022
10. Bustamante, A. L. (2020). Thinger.io plataform. Thinger.Io. https://thinger.io/ Consultado mayo 2022.
11. de Panfilis, G. (2022, 14 febrero). About FIWARE. https://www.fiware.org/about-us/ Consultado: mayo 2022.
12. FIWARE. (2020a). La Plataforma FIWARE - Aprende FIWARE en español. https://fiware-training.readthedocs.io/es_MX/latest/ecosistemaFIWARE/plataformaFIWARE/ Consultado: agosto 2022.
13. Gakenheimer, R. (1998). Los problemas de la movilidad en el mundo en desarrollo. EURE (Santiago), 24(72). https://doi.org/10.4067/s0250-71611998007200002
14. Bull, A. & Cooperation, G. A. F. T. (2003). Congestión de tránsito: el problema y cómo enfrentarlo. CUADERNOS DE LA CEPAL.
15. European Research Cluster on the Internet of Things. (2016). Internet of Things. Coordinating and building a broadly based consensus on the ways to realise the Internet of Things vision in Europe. http://www.internet-of-things-research.eu/about_iot.htm Consultado: septiembre 2022.
16. Holloway, C. (2018, 1 octubre). Plataformas IoT. IT Masters Mag. https://www.itmastersmag.com/noticias-analisis/plataformas-iot-que-son-y-como-elegir-la-mejor-para-el-negocio/ Consultado: agosto 2022.
17. A. (2015, 3 marzo). Telefónica, Orange, Engineering y unen fuerzas para impulsar estándares comunes para Smart Cities basados en la plataforma FIWARE. Atos. https://atos.net/es/2015/comunicados-de-prensa-es_2015_03_03/es-pr-

2015_03_03_01?utm_source=es.atos.net/es-es/home/quienes-somos/noticias-y-eventos/noticias/2015/pr-2015_03_03_01.html&utm_medium=301 Consultado: agosto 2022.
18. Telefónica. (2017, 24 octubre). FIWARE, el estándar que necesita el Internet de las Cosas. Catedra Telefónica de la Universidad de Extremadura. http://catedratelefonica.unex.es/fiware-el-estandar-que-necesita-el-internet-de-las-cosas/#:~:text=FIWARE%20es%20una%20iniciativa%20open,%2C%20Smart%20Factories%2C%20entre%20otros. Consultado: agosto 2022.
19. Martinez, Ramirez, Estrada, Torres, A. F. H. L. A. (2017). GENERIC MODULE FOR COLLECTING DATA IN SMART CITIES. 2nd International Conference on Smart Data and Smart Cities, XLII-4/W3, 66–67.
20. Wences. (2018). Desarrollo de un sistema inteligente de pronóstico de llegada y disponibilidad de asientos de las unidades de transporte público urbano [Tesis de Maestría]. Centro Nacional de Investigación y Desarrollo Tecnológico.
21. FIWARE. (2019). QuantumLeap - QuantumLeap. https://quantumleap.readthedocs.io/en/latest/#overview Consultado: septiembre 2022.
22. FIWARE. (2019b). Modelos de datos. https://fiware-training.readthedocs.io/es_MX/latest/ecosistemaFIWARE/modelosdedatos/#modelos-de-datos-armonizados-de-transporte Consultado: septiembre 2022.
23. FIWARE. (2019c). Vehicle - FIWARE DataModels. https://fiware-datamodels.readthedocs.io/en/latest/Transportation/Vehicle/Vehicle/doc/spec/index.html Consultado septiembre 2022.
24. FIWARE. (2019d). *TrafficFlowObserved* - FIWARE DataModels. https://fiware-datamodels.readthedocs.io/en/latest/Transportation/*TrafficFlowObserved*/doc/spec/index.html Consultado: septiembre 2022.
25. Google Developers. (2021, 10 febrero b). Descripción general de la especificación GTFS estática. https://developers.google.com/transit/gtfs?hl=es
26. HERE Technologies. (2022). HERE | About Us. HERE Maps. https://www.here.com/company/about-us Consultado: septiembre 2022.
27. HERE Maps. (2022 b). Build apps with HERE Maps API and SDK Platform Access - HERE Traffic API. HERE Developer. https://developer.here.com/documentation/examples/rest/traffic/traffic-flow-bounding-box Consultado: septiembre 2022.
28. Verdezoto, T. Z. A. (2020). Análisis del congestionamiento vehicular para el mejoramiento de vía principal en Guayaquil-Ecuador. https://www.redalyc.org/journal/5703/570363740001/html/ Consultado: septiembre 2022.
29. Wang, Y., König, L. M., & Reiterer, H. (2021). A smartphone app to support sedentary behavior change by visualizing personal mobility patterns and action planning (SedVis): Development and pilot study. JMIR Formative Research, 5(1), e15369.
30. Martins, T. G., Lago, N., De Souza, H. A., Santana, E. F. Z., Telea, A., & Kon, F. (2020). Visualizing the structure of urban mobility with bundling: A case study of the city of São Paulo. Anais do Workshop de Computação Urbana (CoUrb 2020). Sociedade Brasileira de Computação - SBC.
31. Zeng, W., Fu, C.-W., Arizona, S. M., Erath, A., & Qu, H. (2014). Visualizing mobility of public transportation system. IEEE Transactions on Visualization and Computer Graphics, 20(12), 1833–1842

32. Alemany, J. L. P. (2019). Creación de un repositorio de contenidos para transporte público urbano con Fiware. Universidad Politécnica de Valencia Escuela Técnica Superior de Ingeniería Geodésica, Cartográfica y Topografía, Valencia.
33. Huang, X., Li, Z., Jiang, Y., Ye, X., Deng, C., Zhang, J., & Li, X. (2020). The characteristics of multi-source mobility datasets and how they reveal the luxury nature of social distancing in the U.S. during the COVID-19 pandemic. doi:10.1101/2020.07.31.20143016
34. Jeffrey, B., Walters, C. E., Ainslie, K. E. C., Eales, O., Ciavarella, C., Bhatia, S., … Riley, S. (2020). Anonymised and aggregated crowd level mobility data from mobile phones suggests that initial compliance with COVID-19 social distancing interventions was high and geographically consistent across the UK. Wellcome Open Research, 5, 170.
35. FIWARE. (s. f.). GitHub - FIWARE/tutorials.Time-Series-Data: FIWARE 304: Querying Time Series Data (QuantumLeap). GitHub. https://github.com/FIWARE/tutorials.Time-Series-Data

Anexo A

Implementación de un servidor FIWARE

A. Implementación de un servidor con las funcionalidades de FIWARE

Inicialmente, el servidor donde se implementarán los componentes de FIWARE debe de contar con sistema operativo basado en Linux, en nuestro caso Xubuntu en su última versión. Posteriormente, en la Figura A1 se muestra los componentes necesarios para utilizar la solución Quantumleap de la Plataforma FIWARE, asimismo se muestran los puertos de comunicación que utilizan entre los componentes.

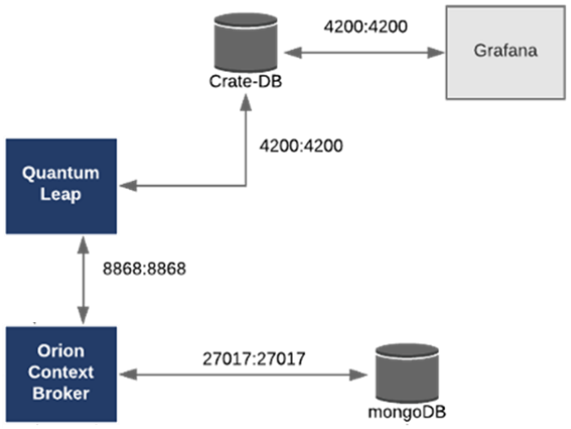


Figura A1: Componentes principales de Quantumleap

Para la implementación de estos componentes en un servidor con sistema operativo Xubuntu se utiliza la herramienta Docker. Esta herramienta como su nombre lo indica es un contenedor de aplicaciones, cada contenedor con una aplicación se le conoce como imagen de contendor Docker. Esta imagen es un paquete de software ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación específica.

Los contenedores y las máquinas virtuales tienen beneficios similares, ya que aíslan y asignan recursos computaciones. Pero funcionan de manera diferente, esto debido a que los contenedores virtualizan el sistema operativo y las máquinas virtuales el hardware. Docker contendrá una imagen por cada componente que se presenta en la Figura A1.

El proceso de instalación de la herramienta Docker y Docker-compose se muestra a continuación:

A.1 Instalación de Docker

1) Se desinstala la versión que tiene por defecto el sistema operativo.

Comando: $ sudo apt-get remove docker docker-engine docker.io containerd runc

2) Se actualizan los índices de paquetes del sistema operativo.

Comando: $ sudo apt-get update

3) Se instalan los paquetes para permitir el uso de un repositorio a través del protocolo HTTPS.

Comando: $ sudo apt-get install \ ca-certificates \ curl \ gnupg \ lsb-release

4) Se agregan las claves GPG oficiales del Docker.

Comando 1: $ sudo mkdir -p /etc/apt/keyrings

Comando 2: $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

5) Se configura el repositorio estable de Docker.

Comando: sudo echo \ "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \ $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

6) Se instala la versión más reciente de Docker.

Comando: $ sudo apt-get install docker-ce docker-ce-cli containerd.io

7) Validation de la instalación.

Se consulta la versión de Docker instalado.

Comando: sudo docker –version

Respuesta: Docker version 20.10.19, build d85ef84

A.2 Instalación de docker-compose

1) Descargar la versión estable de Docker Compose.

Comando: $ sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

2) Aplicar los permisos de ejecución al archivo de binario.

Comando: $ sudo chmod +x /usr/local/bin/docker-compose

3) Validación de la instalación.

Se consulta la versión de Docker Compose instalado.

Comando: sudo docker-compose –version

Resultado: docker-compose version 1.29.2, build 5becea4c

A.3 Despliegue de los componentes de FIWARE

Para el despliegue de los contenedores con las funcionalidades de un servidor FIWARE se utiliza la herramienta Docker y Docker-Compose. Estas herramientas permiten crear e intercomunicar contenedores con cada uno de los servicios que se muestran en la Figura A1 dentro de un mismo Sistema Operativo.

Para lograr el despliegue de los contenedores con las herramientas de Docker se obtiene un archivo con extensión YML. Este archivo contiene las instrucciones necesarias para la instalación y configuración de cada uno de los contenedores. En seguida se muestra el procedimiento para el despliegue de los contendores.

1) Descargar el docker-compose.yml donde se encuentra las instrucciones de instalación y configuración de los contenedores.

https://github.com/dSalvoG/ML-Urban-Sound-Classifier/blob/master/docker/docker-compose.yml

2) Aumentar el número máximo de mapas de memoria que pueden tener los procesos de los contenedores.

Comando: $ sudo sysctl -w vm.max_map_count=262144

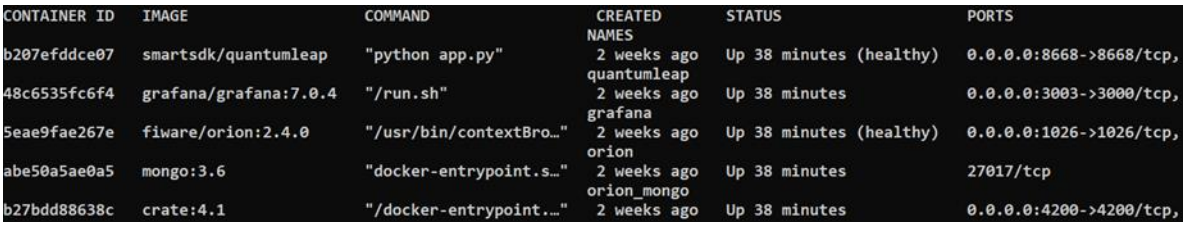
3) Abrir una terminal con la ubicación donde se encuentra el archivo docker-compose.yml y ejecutar el archivo con la herramienta Docker Compose.

Comando: $ sudo docker-compose -f docker-compose.yml up –d

4) Validación de la creación de los contenedores.

Comando: $ sudo docker ps

El resultado de la validación de la creación de los contenedores se puede observar en la Figura A2.



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b207efddce07	smartsdk/quantumleap	"python app.py"	2 weeks ago	Up 38 minutes (healthy)	0.0.0.0:8668->8668/tcp,	quantumleap
48c6535fc6f4	grafana/grafana:7.0.4	"/run.sh"	2 weeks ago	Up 38 minutes	0.0.0.0:3003->3000/tcp,	grafana
5eae9fae267e	fiware/orion:2.4.0	"/usr/bin/contextBro…"	2 weeks ago	Up 38 minutes (healthy)	0.0.0.0:1026->1026/tcp,	orion
abe50a5ae0a5	mongo:3.6	"docker-entrypoint.s…"	2 weeks ago	Up 38 minutes	27017/tcp	orion_mongo
b27bdd88638c	crate:4.1	"/docker-entrypoint.…"	2 weeks ago	Up 38 minutes	0.0.0.0:4200->4200/tcp,	

Figura A2: Contenedores de los servicios de FIWARE

De igual manera se prueba el acceso mediante el navegador web.

Liga CreateDB: http://ipServer:4200/

Liga *Orion Context Broker*: http://ipServer:1026/version

Liga Grafana: http://ipServer:3003/

Liga Quantumleap: http://ipServer:8668/version

Anexo B

Alimentadores ADMU desarrollados

B.1 Archivo de configuración para el despliegue del servidor FIWARE

Se muestra el archivo de configuración necesario para el despliegue de los contenedores correspondientes a las funcionalidades de FIWARE. Desde una terminal con permisos de administrados se ejecuta el comando:

Comando: sudo docker-compose -f docker-compose-fiware.yml up -d

Docker-compose-FIWARE.yml

```
version: '3.5'

services:
  # Orion Context Broker
  orion:
    image: fiware/orion:2.4.0
    hostname: orion
    container_name: orion
    depends_on:
      - orion_mongo
    networks:
      - fiware_orion_net
    ports:
      - "1026:1026"
    command: -dbhost orion_mongo -noCache
    healthcheck:
      test: curl --fail -s http://127.0.0.1:1026/version || exit 1

  # Monitorizacion utilizando Grafana
  grafana:
    image: grafana/grafana:7.0.4
    container_name: grafana
    depends_on:
        - cratedb
    ports:
        - "3003:3000"
    environment:
        - GF_INSTALL_PLUGINS=https://github.com/orchestracities/grafana-map-
plugin/archive/master.zip;grafana-map-plugin,grafana-clock-panel,grafana-worldmap-panel
    networks:
      - fiware_grafana_net
    volumes:
      - type: volume
        source: grafana_storage
        target: /var/lib/grafana

  # Quantum Leap is persisting Short Term History to Crate-DB
```

```
quantumleap:
  image: smartsdk/quantumleap
  hostname: quantumleap
  container_name: quantumleap
  ports:
    - "8668:8668"
  networks:
    - fiware_grafana_net
    - fiware_orion_net
  depends_on:
    - cratedb
    - orion
    - orion_mongo
  environment:
    - "CRATE_HOST=http://cratedb"
  healthcheck:
    test: curl --fail -s http://127.0.0.1:8668/version || exit 1

# MONGO DB solo para servicio de Orion
orion_mongo:
  image: mongo:3.6
  container_name: orion_mongo
  hostname: orion_mongo
  expose:
    - "27017"
  volumes:
    - type: volume
      source: orion_mongo_db
      target: /data/db
      read_only: false
    - type: volume
      source: orion_mongo_configdb
      target: /data/configdb
      read_only: false
  networks:
    - fiware_orion_net

# Crate para utilizar Grafana
cratedb:
  image: crate:4.1
  hostname: cratedb
  container_name: cratedb
  ports:
    # Admin UI
    - "4200:4200"
```

```
      # Transport protocol
      - "4300:4300"
      - "5432:5432"
   # command: crate -Clicense.enterprise=false -Cauth.host_based.enabled=false  -
Ccluster.name=democluster -Chttp.cors.enabled=true -Chttp.cors.allow-origin="*"
    networks:
      - fiware_grafana_net
    command: -Cdiscovery.type=single-node -Chttp.cors.enabled=true -Chttp.cors.allow-
origin="*"
    volumes:
      - type: volume
        source: crate_db_storage
        target: /data

# Redes
networks:
  fiware_orion_net:
  fiware_grafana_net:

# Volumenes para persistir la data.
volumes:
  orion_mongo_db:
  orion_mongo_configdb:
  crate_db_storage:
  grafana_storage:
```

B.2 Alimentador ADMU para mapear datos de los Metrobuses de la CDMX

Se muestra el código completo del alimentado ADMU que se encarga de estandarizados los datos de los Metrobuses de la Ciudad de México. Los archivos son: suscripciónVehicle.json, funcionConsultaDatosGTFS.py, funcionLlenarJSON.py, alimentador_GTFS_aFIWARE_vehicle.py

Con ayuda de una herramienta que realice peticiones REST (en el caso de este proyecto de tesis la herramienta Postman) se envía la petición POST con el contenido del archivo suscripciónVehicle.json.

Para el funcionamiento del alimentador ADMU para mapear datos de los Metrobuses de la CDMX, los archivos con extensión Python deben de estar en un mismo directorio y ejecutar únicamente el denominado alimentador_GTFS_aFIWARE_vehicle.py.

Suscripción de la entidad *Vehicle*
suscripciónVehicle.json

```
{
    "description":"Notificación Quantumleap para Vehiculo",
    "subject":{
        "entities":[{
            "idPattern":"Vehicle:MetrobusCDMX:*",
            "type":Vehicle
        }
        ],
        "condition":{
            "attrs":[
                "dateModified"
            ]
        }
    },
    "notification":{
        "attrs":[
            "location",
            "serviceStatus",
            "speed",
            "heading",
            "mileageFromOdometer",
            "dateModified",
            "dateObserved"
        ],
        "http":{
            "url":"http://quantumleap:8668/v2/notify"
        },
        "metadata":[
            "dateCreated",
```

```
            "dateModified"
        ]
    }
}
```

Función para extracción de los datos de los Metrobuses
funcionConsultaDatosGTFS.py

```
#Paqueterias de python para utilizar datos GTFS-RT y request
from google.transit import gtfs_realtime_pb2
import requests

def consumeGTFS():
    feed = gtfs_realtime_pb2.FeedMessage()
    # petición request hacia la liga http://app.citi-mb.mx/GTFS-RT/vehiculosPosicion
    user='MB456E;V'
    pss='NCKQONZRLW'
    response = requests.get('http://'+user+':'+pss+'@app.citi-mb.mx/GTFS-
RT/vehiculosPosicion')
    feed.ParseFromString(response.content)

    return feed
```

Funciones donde se encuentras las estructuras JSON del modelo de datos *Vehicle*
funcionLlenarJSON.py

```
import json
def llenaJSONactualizacion(latitudeVehicle,
                           longitudeVehicle,
                           timeConsulta,
                           statusVehicle,
                           speedVehicle,
                           bearingVehicle,
                           odometerVehicle):
    data={
    "dateObserved": {
            "type": "DateTime",
            "value": timeConsulta
        },
        "dateModified": {
            "type": "DateTime",
            "value": timeConsulta
        },
        "location": {
            "type": "geo:json",
            "value": {
                "type": "Point",
                "coordinates": [
```

```
                longitudeVehicle,
                latitudeVehicle]
        },
        "metadata": {
            "timestamp": {
                "type": "DateTime",
                "value": timeConsulta
            }
        }
    },
    "serviceStatus": {
        "value": statusVehicle
    },
    "speed": {
        "value": speedVehicle,
        "metadata": {
            "timestamp": {
                "type": "DateTime",
                "value": timeConsulta
            }
        }
    },
    "heading": {
        "value": bearingVehicle,
        "metadata": {
            "timestamp": {
                "type": "DateTime",
                "value": timeConsulta
            }
        }
    },##Se agrega odometro
    "mileageFromOdometer": {
        "value": odometerVehicle,
        "metadata": {
            "timestamp": {
                "type": "DateTime",
                "value": timeConsulta
            }
        }
    }
}

payload = json.dumps(data)

return payload
```

```
def llenaJSONcreacionEntidad(labelVehicle,
                             latitudeVehicle,
                             longitudeVehicle,
                             timeConsulta,
                             statusVehicle,
                             speedVehicle,
                             bearingVehicle,
                             odometerVehicle):
    data={
        "id":
"Vehicle:MetrobusCDMX:"+labelVehicle,

        "type": Vehicle,
        "dateObserved": {
            "type": "DateTime",
            "value": str(timeConsulta)
    },
    "dateModified": {
            "type": "DateTime",
            "value": str(timeConsulta)
    },
        "category": {
            "value": [
                "public"
            ]
        },
        "vehicleType": {
            "value": "bus"
        },
        "fleetVehicleId": {
            "value": labelVehicle
        },
        "location": {
            "type": "geo:json",
            "value": {
                "type": "Point",
                "coordinates": [
                    longitudeVehicle,
                    latitudeVehicle
                ]
            },
            "metadata": {
                "timestamp": {
                    "type": "DateTime",
```

```
                    "value": timeConsulta
                }
            }
        },
        "serviceStatus": {
            "value": str(statusVehicle)
        },
        "speed": {
            "value": speedVehicle,
            "metadata": {
                "timestamp": {
                    "type": "DateTime",
                    "value": timeConsulta
                }
            }
        },
        "heading": {
            "value": bearingVehicle,
            "metadata": {
                "timestamp": {
                    "type": "DateTime",
                    "value": timeConsulta
                }
            }
        },
        "mileageFromOdometer": {
            "value": odometerVehicle,
            "metadata": {
                "timestamp": {
                    "type": "DateTime",
                    "value": timeConsulta
                }
            }
        }
    }

    payload = json.dumps(data)

    return payload
```

Código que identifica, mapea y envía los datos de congestión vehicular al servidor FIWARE alimentador_GTFS_aFIWARE_vehicle.py

```
from funcionLlenarJSON import llenaJSONactualizacion, llenaJSONcreacionEntidad
from funcionConsultaDatosGTFS import consumeGTFS
import requests
```

```
from datetime import datetime
import time

#Cambiar IP de acuerdo a la IP que se le asigne
ipServer = '127.0.0.1'

headers_string={
        'Content-Type':'application/json'
        }

while True:
    try:
        #Datos GTFS-RT dentro del Feed (datos de todos los Metrobuses)
        feed=consumeGTFS()
        #Unicia el contador el cual recorre cada entidad dentro de GTFS-RT
        i=0
        #Acceso a los datos por vehiculo, conservando el valor de los atributos que se
necesitan
        for i in range(len(feed.entity)):
            Metrobus=feed.entity[i]
            idVehicle=Metrobus.vehicle.vehicle.id
            labelVehicle=Metrobus.vehicle.vehicle.label
            latitudeVehicle=Metrobus.vehicle.position.latitude
            longitudeVehicle=Metrobus.vehicle.position.longitude
            #tranformación del dato TIMESTAMP
            timeConsulta=datetime.fromtimestamp(Metrobus.vehicle.timestamp).isoformat()
            #transformación del dato CURRENT_STATUS
            if (Metrobus.vehicle.current_status == 2):
                statusVehicle="onRute,IN_TRANSIT_TO"
            else:
                statusVehicle="onRute,STOPPED_AT"
            #transformación del dato SPEED
            speedVehicle=Metrobus.vehicle.position.speed*3.6
            bearingVehicle=Metrobus.vehicle.position.bearing
            #tranformación del dato ODOMETER
            odometerVehicle=Metrobus.vehicle.position.odometer/1000

            #URL para creación de entidades
            urlCreacionEntidad = "http://"+ str(ipServer) +":1026/v2/entities/"
            payloadCreacionEntidad=llenaJSONcreacionEntidad(labelVehicle,
                                                            latitudeVehicle,
                                                            longitudeVehicle,
                                                            timeConsulta,
                                                            statusVehicle,
                                                            speedVehicle,
```

```
                                                        bearingVehicle,
                                                        odometerVehicle)

            estatusResquestPOST = requests.post(url= urlCreacionEntidad,
                                                data=payloadCreacionEntidad,
                                                headers= headers_string)
            #imprime el estado del POST
            print("VEHICLE Estado POST: {}" .format(estatusResquestPOST.status_code))

            if(estatusResquestPOST.status_code==422):
                #URL para actualización de
entidades
       #Se cambia idVehicle por labelVehicle
                urlActualizacionEntidad = "http://"+ str(ipServer)
+":1026/v2/entities/Vehicle:MetrobusCDMX:"+str(labelVehicle)+"/attrs"
                payloadActualizacionEntidad=llenaJSONactualizacion(latitudeVehicle,
                                                                   longitudeVehicle,
                                                                   timeConsulta,
                                                                   statusVehicle,
                                                                   speedVehicle,
                                                                   bearingVehicle,
                                                                   odometerVehicle)
                estatusResquestPACH = requests.patch(url= urlActualizacionEntidad,
                                                     data=payloadActualizacionEntidad,
                                                     headers= headers_string)
                #imprime el estado del PACH
                print("VEHICLE Estado PACH: {}"
.format(estatusResquestPACH.status_code))
    except requests.exceptions.RequestException:
        print("VEHICLE Error de Conexion")
        time.sleep(10)
```

B.3 Alimentador ADMU para mapear datos de congestión vehicular de la CDMX

Se muestra el código completo del alimentado ADMU que se encarga de estandarizados los datos de congestión vehicular de la Ciudad de México. Los archivos son: suscripción*TrafficFlowObserved*.json, funcionConsumirDatosHereMaps.py, funcionLlenarJSONTrafico.py, funcionesTransformacionDatosHereMaps.py, alimentador_GTFS_aFIWARE_transitFlowObserved.py

Con ayuda de una herramienta que realice peticiones REST (en el caso de este proyecto de tesis la herramienta Postman) se envía la petición POST con el contenido del archivo suscripción*TrafficFlowObserved*.json.

Para el funcionamiento del alimentador ADMU para mapear datos de la congestión vehicular de la CDMX, los archivos con extensión Python deben de estar en un mismo directorio y ejecutar únicamente el denominado alimentador_GTFS_aFIWARE_transitFlowObserved.py.

Suscripción de la entidad *TrafficFlowObserved*
suscripción*TrafficFlowObserved*.json

```
{
    "description":"Notificación Quantumleap para Trafico",
    "subject":{
        "entities":[{
            "idPattern":"TrafficFlowObserved:*",
            "type":TrafficFlowObserved
        }
        ],
        "condition":{
            "attrs":[
                "dateModified"
            ]
        }
    },
    "notification":{
        "attrs":[
            "dateObserved",
            "dateModified",
            "occupancy",
            "averageVehicleSpeed"
        ],
        "http":{
            "url":"http://quantumleap:8668/v2/notify"
        },
        "metadata":[
            "dateCreated",
            "dateModified"
```

```
        ]
    }
}
```

Función para extracción de los datos de congestión vehicular de los segmentos de la CDMX
funcionConsumirDatosHereMaps.py

```
import requests

def consumeHereMaps():
    coordenadasBbox = '19.725181,-99.316750;19.061956,-98.788376'
    apiKey= 'dvK17Bzm7_rtFgRUIKH_BvZOcOdmrBDC0H7GiapjlWE'

    url = 'https://traffic.ls.hereapi.com/traffic/6.2/flow.json?bbox='+
str(coordenadasBbox) +'&responseattributes=sh&apiKey='+str(apiKey)
    dataHere = requests.get(url)
    dataHereJson = dataHere.json()

    return dataHereJson
```

Funciones donde se encuentras las estructuras JSON del modelo de datos *TrafficFlowObserved*
funcionLlenarJSONTrafico.py

```
import json

def llenaJSONcreacionEntidadTrafico(TMC_PC, TMC_QD, TMC_DE, CF_SP, CF_JF, SHP,
horaConsulta):
    data={
        "id": "TrafficFlowObserved:CDMX:"+str(TMC_PC)+str(TMC_QD),
                "type": TrafficFlowObserved,
                "dateObserved": {
                    "type": "DateTime",
                    "value": str(horaConsulta)
                },
                "dateModified": {
                    "type": "DateTime",
                    "value": str(horaConsulta)
                },
                "occupancy": {
                    "value": float(CF_JF)
                },
                "address": {
                    "type": "StructuredValue",
                    "value": {
                        "addressLocality": "Ciudad de México",
                        "addressCountry": "MX",
                        "streetAddress": str(TMC_DE)
```

```
                    }
                },
                "averageVehicleSpeed": {
                    "value": float(CF_SP)
                },
                "location": {
                    "type": "geo:json",
                    "value": {
                        "type": "MultiLineString",
                        "coordinates": SHP
                    }
                }
        }
    payload = json.dumps(data)

    return payload

def llenaJSONactualizacionEntidadTrafico(CF_SP, CF_JF, horaConsulta):


    data={
        "dateObserved": {
            "type": "DateTime",
            "value": horaConsulta
        },
        "dateModified": {
            "type": "DateTime",
            "value": horaConsulta
        },
        "occupancy": {
            "value": float(CF_JF)
        },
        "averageVehicleSpeed": {
            "value": float(CF_SP)
        }
    }

    payload = json.dumps(data)

    return payload
```

Funciones para la trasformación de atributos
funcionesTransformacionDatosHereMaps.py

```
#Reescalar el valor CF_JF
def rescale(X,A,B,C,D,force_float=True): #val, minOld, maxOld, minNew, maxNew
```

```
    retval = ((float(X - A) / (B - A)) * (D - C)) + C
    if not force_float and all(map(lambda x: type(x) == int, [X,A,B,C,D])):
        return int(round(retval))
    return retval

#Limpia caracteres para TMC_DE
def limpiaTexto (texto):
    caracter={"(",")",";","=","'","\"",">","<"}
    for i in caracter:
      texto=texto.replace(i, '')
    return texto

#Transformación de un conjunto de coordenadas de HereMpas a un MultiLineString de JSON
def getCoordinates(shape):
    coordinates =[]
    for SHP in shape:
        for coord in SHP['value']:
          innerCoord=[]
          for val in coord.rstrip().split(' '):
            latlong = val.split(',')
            innerCoord.append([float(latlong[1]),float(latlong[0])])
          coordinates.append(innerCoord)
    return coordinates

#Se realiza la transformación de la hora acorde a la zona horaria
import datetime
from dateutil import tz
def cambiaZonaHoraria(horaUTC):
    from_zone = tz.gettz('UTC')
    to_zone = tz.gettz('America/Mexico_City')
    fechaUTC0 = horaUTC
    utc = datetime.datetime.strptime(fechaUTC0, "%Y-%m-%dT%H:%M:%S.%f%z")
    utc = utc.replace(tzinfo=from_zone)
    cst = utc.astimezone(to_zone).strftime('%Y-%m-%dT%H:%M:%S')
    #print(utc)
    #print(cst)
    return cst
```

Código que identifica, mapea y envía los datos de congestión vehicular al servidor FIWARE alimentador_GTFS_aFIWARE_transitFlowObserved.py

```
from funcionConsumirDatosHereMaps import consumeHereMaps
from funcionesTranformacionDatosHereMaps import rescale, limpiaTexto, getCoordinates, cambiaZonaHoraria
from funcionLlenarJSONTrafico import llenaJSONcreacionEntidadTrafico, llenaJSONactualizacionEntidadTrafico
import requests
import time
```

```
#Cambiar IP de acuerdo a la IP que se le asigne
ipAddressServer = '127.0.0.1'

headers_string={
        'Content-Type':'application/json'
        }

while True:
    try:
        #Función que consulta los datos de HERE Maps
        dataHereJson=consumeHereMaps()
        #Recorrido que se hace a todo el JSON
        for RWS in range(len(dataHereJson['RWS'])):
          for RW in range(len(dataHereJson['RWS'][RWS]['RW'])):
            for FIS in range(len(dataHereJson['RWS'][RWS]['RW'][RW]["FIS"])):
              for FI in range(len(dataHereJson['RWS'][RWS]['RW'][RW]["FIS"][FIS]['FI'])):
                datos = dataHereJson['RWS'][RWS]['RW'][RW]['FIS'][FIS]['FI'][FI]
                TMC_PC=datos['TMC']['PC']
                TMC_QD=datos['TMC']['QD']
                #Limpia el texto de los simbolos especificos
                TMC_DE=limpiaTexto(datos['TMC']['DE'])
                CF_SP=datos['CF'][0]['SP']
                #Re-escala el valor
                CF_JF=rescale(float(datos['CF'][0]['JF']),0,10,0,1)
                #Tranforma el conjunto de coordenadas en un geoJSON MultiLineString
                SHP=getCoordinates(datos['SHP'])
                horaConsulta=cambiaZonaHoraria(dataHereJson['CREATED_TIMESTAMP'])

                #URL para creación de entidades
                urlCreacionEntidadTrafico = "http://"+ str(ipAddressServer) +":1026/v2/entities/"
                payloadCreacionEntidadTrafico=llenaJSONcreacionEntidadTrafico(TMC_PC,
                                                                              TMC_QD,
                                                                              TMC_DE,
                                                                              CF_SP,
                                                                              CF_JF,
                                                                              SHP,
                                                                              horaConsulta)

                estatusResquestPOSTTrafico = requests.post(url= urlCreacionEntidadTrafico,
                                                           data=payloadCreacionEntidadTrafico,
                                                           headers= headers_string)

                #imprime el estado del POST
                print("TRAFFIC Estado POST: {}" .format(estatusResquestPOSTTrafico.status_code))
```

```
                if(estatusResquestPOSTTrafico.status_code==422):
                    #URL para actualización de entidades
                    urlActualizacionEntidadTrafico = "http://"+ str(ipAddressServer)
+":1026/v2/entities/TrafficFlowObserved:CDMX:"+str(TMC_PC)+str(TMC_QD)+"/attrs"
                    payloadActualizacionEntidadTrafico = llenaJSONactualizacionEntidadTrafico(CF_SP,
                                                                                              CF_JF,
                                                                                              horaConsulta)
                    estatusResquestPACHTrafico = requests.patch(url= urlActualizacionEntidadTrafico,
                                                                data=payloadActualizacionEntidadTrafico,
                                                                headers= headers_string)
                    #imprime el estado del PACH
                    print("TRAFFIC Estado PACH: {}" .format(estatusResquestPACHTrafico.status_code))
    except requests.exceptions.RequestException:
        print("TRAFFIC Error de Conexion")
        time.sleep(10)
```

B.4 Aplicaciones WEB demo (Mapas)

Se muestran los códigos que fueron utilizados para la representación en un mapa con los datos que están alojados en el espacio unificado. Estos archivos pueden ser ejecutados con ayuda de la herramienta XAMP.

Mapa que representa la ubicación de los Metrobuses y congestión vehicular de la Ciudad de México. Ambas representaciones son en tiempo real.
mapaDemoV1.php

```
<!DOCTYPE html>
<html>
<head><title>Mapa Demo V1</title>
<h3>Prueba del mapa, muestra trafico y Metrobuses</h3>
    <meta charset="UTF-8" />

    <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"integrity="sha512-
XQoYMqMTK8LvdxXYG3nZ448hOEQiglfqkJs1NOQV44cWnUrBc8PkAOcXy20w0vlaXaVUearIOBhiXZ5V3ynxwA=="crossori
gin=""></script>

    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
integrity="sha512-
xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZMZ19scR4PsZChSR7A=="crossori
gin=""/>

    <script src="https://code.jquery.com/jquery-3.5.1.js"></script>

</head>
<body>
    <!-- div para el tamaño del mapa -->
    <div id="map" style="width: 1850px; height: 900px"></div>

    <!-- script para dibujar el mapa -->
    <script>
        //var map=L.map('map',{[41.66,-4.72],zoom:10});
        var map = L.map('map').setView([19.427591, -99.132965],15);
        var layer=new L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png');
        map.addLayer(layer);
    </script>

    <!-- Script para dibujar Metrobuses -->
    <script>
        //Propiedades de la linea que se va a pintar
        var customIcon = new L.Icon({
            iconUrl: 'autobusTRES.png',
            iconSize: [35, 35],
            iconAnchor: [20, 35]
```

```
        });

        //Consulta de coordenadas y dibujo de calles
        function pintaMetrobuses(){
            $(".leaflet-marker-icon").remove();
            $.ajax({
                url: "http://172.16.10.220:1026/v2/entities?type=Vehicle&limit=1000",
                type: "GET",
                dateType: "json",
            }).done(function(data){
                $.each(data, function(key, datos){
                    //console.log(datos)
                    coordenadasMetrobus = datos.location.value.coordinates
                    console.log(coordenadasMetrobus)

                    var marker = L.marker([coordenadasMetrobus[1], coordenadasMetrobus[0]],
{icon: customIcon, title:"Metrobus número: " + datos.fleetVehicleId.value + "\nServiceType: " +
datos.vehicleType.value +", "+ datos.category.value + "\nHeading: " + datos.heading.value +
"\nCoordinates (lat,lon): " + datos.location.value.coordinates[1] + ", "+
datos.location.value.coordinates[0] + "\nOdometer: " + datos.mileageFromOdometer.value +
"\nServiceStatus: " + datos.serviceStatus.value + "\nSpeed(KM/H): " + datos.speed.value +
"\nTime: " + datos.dateModified.value});

                    marker.addTo(map);
                })

            })
        }
        setInterval('pintaMetrobuses()',15000);
    </script>

    <!-- Script para dibujar calles -->
    <script>
        //Propiedades de la linea que se va a pintar
        var alto = {
            "color": "#8a030a",
            "weight": 5,
            "opacity": 0.70
        };
        var medio = {
            "color": "#b9770e",
            "weight": 5,
            "opacity": 0.70
        };
        var bajo = {
```

```
            "color": "#04b316",
            "weight": 5,
            "opacity": 0.70
        };

        //Consulta de coordenadas y dibujo de calles
        function pintaOcupacion(liga){
            $(".leaflet-interactive").remove();
            $(".leaflet-popup").remove();
            $.ajax({
                url: liga,
                type: "GET",
                dateType: "json",
            }).done(function(data){
                $.each(data, function(key, datos){
                    //Obtiene las corrdenadas en formato geoJSON
                    coordenadaCalles = datos.location.value
                    //Obtiene la ocupación para la condición del color de la linea
                    ocupacion = datos.occupancy.value
                    //console.log(datos)
                    if(ocupacion <= 0.3){
                    var myLayer = L.geoJSON(coordenadaCalles, {style: bajo}).addTo(map);
                    }else if(ocupacion >0.3 && ocupacion <= 0.7){
                        var myLayer = L.geoJSON(coordenadaCalles, {style: medio}).addTo(map);
                    }else{
                        var myLayer = L.geoJSON(coordenadaCalles, {style: alto}).addTo(map);
                    }

                    idCalle = datos.id
                    direccion = datos.address.value.streetAddress
                    localidad = datos.address.value.addressLocality
                    pais = datos.address.value.addressCountry
                    velocidad = datos.averageVehicleSpeed.value
                    horaObservacion = datos.dateObserved.value
                    porcentajeOcupacion = ocupacion*100

                    //console.log(velocidad)
                    myLayer.bindPopup('<h3>Información de Trafico</h3><p>ID: '+ idCalle
+'<br>Dirección: '+ direccion +', '+ localidad +', '+ pais +'<br>Velocidad: ' + velocidad + '
Km/h<br>Porcentaje Ocupación: '+ porcentajeOcupacion.toFixed(1) +'%<br>hora: ' +
horaObservacion);
                })

            })
        }
```

```
        //Pinta la ocupación de las calles, avenidas, bulevares en el mapa, se generan 5 ligas debido a que FIWARA se puede copnsumir entidades de 1000 en 1000 -- 240000 = a 4min
        setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=0")',60000); //1000
        setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=1000")',60000); //1000
        setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=2000")',60000);//1000
        setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=3000")',60000); //1000
        setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=4000")',60000) //132
    </script>

</body>
</html>
```

Mapa que representa únicamente la ubicación en tiempo real de los Metrobuses de la Ciudad de México.
mapaMetrobus.php

```
<!DOCTYPE html>
<html>
<head><title>Metrobuses Prueba</title>
<h3>Mapa Metrobuses CDMX</h3>
    <meta charset="UTF-8" />

    <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"integrity="sha512-XQoYMqMTK8LvdxXYG3nZ448hOEQiglfqkJs1NOQV44cWnUrBc8PkAOcXy20w0vlaXaVUearIOBhiXZ5V3ynxwA==" crossorigin=""></script>

    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" integrity="sha512-xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZMZ19scR4PsZChSR7A==" crossorigin=""/>

    <script src="https://code.jquery.com/jquery-3.5.1.js"></script>

</head>
<body>
    <!-- div para el tamaño del mapa -->
    <div id="map" style="width: 1850px; height: 900px"></div>

    <!-- script para dibujar el mapa -->
    <script>
```

```
        //var map=L.map('map',{[41.66,-4.72],zoom:10});
        var map = L.map('map').setView([19.427591, -99.132965],15);
        var layer=new L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png');
        map.addLayer(layer);
    </script>

    <!-- Script para dibujar calles -->
    <script>
        //Propiedades de la linea que se va a pintar
        var customIcon = new L.Icon({
            iconUrl: 'bus_icon3.png',
            iconSize: [35, 35],
            iconAnchor: [20, 35]
        });

        //Consulta de coordenadas y dibujo de calles
        function pintaMetrobuses(){
            $(".leaflet-marker-icon").remove();
            $.ajax({
                url: "http://172.16.10.220:1026/v2/entities?type=Vehicle&limit=1000&offset=0",
                type: "GET",
                dateType: "json",
            }).done(function(data){
                $.each(data, function(key, datos){
                    //console.log(datos)
                    coordenadasMetrobus = datos.location.value.coordinates
                    console.log(coordenadasMetrobús)

                    var marker = L.marker([coordenadasMetrobus[1], coordenadasMetrobus[0]],
{icon: customIcon, title:"Metrobus número: " + datos.fleetVehicleId.value + "\nServiceType: " +
datos.vehicleType.value +", "+ datos.category.value + "\nHeading: " + datos.heading.value +
"\nCoordinates (lat,lon): " + datos.location.value.coordinates[1] + ", "+
datos.location.value.coordinates[0] + "\nOdometer: " + datos.mileageFromOdometer.value +
"\nServiceStatus: " + datos.serviceStatus.value + "\nSpeed(KM/H): " + datos.speed.value +
"\nTime: " + datos.dateModified.value});

                    marker.addTo(map);
                })

            })
        }
        setInterval('pintaMetrobuses()',15000);
    </script>
</body>
</html>
```

Mapa que representa únicamente la congestión vehicular en tiempo real de la Ciudad de México.
mapaTraficoOcupacidad.php

```
<!DOCTYPE html>
<html>
<head><title>Trafico Prueba</title>
<h3>Mapa Trafico Pruebas</h3>
    <meta charset="UTF-8" />

    <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"integrity="sha512-
XQoYMqMTK8LvdxXYG3nZ448hOEQiglfqkJs1NOQV44cWnUrBc8PkAOcXy20w0vlaXaVUearIOBhiXZ5V3ynxwA=="crossori
gin=""></script>

    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
integrity="sha512-
xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZMZ19scR4PsZChSR7A=="crossori
gin=""/>

    <script src="https://code.jquery.com/jquery-3.5.1.js"></script>

</head>
<body>
    <!-- div para el tamaño del mapa -->
    <div id="map" style="width: 1850px; height: 900px"></div>

    <!-- script para dibujar el mapa -->
    <script>
        //var map=L.map('map',{[41.66,-4.72],zoom:10});
        var map = L.map('map').setView([19.427591, -99.132965],15);
        var layer=new L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png');
        map.addLayer(layer);
    </script>

    <!-- Script para dibujar calles -->
    <script>
        //Propiedades de la linea que se va a pintar
        var alto = {
            "color": "#8a030a",
            "weight": 5,
            "opacity": 0.70
        };
        var medio = {
            "color": "#b9770e",
            "weight": 5,
```

```
            "opacity": 0.70
        };
        var bajo = {
            "color": "#04b316",
            "weight": 5,
            "opacity": 0.70
        };

        //Consulta de coordenadas y dibujo de calles
        function pintaOcupacion(liga){
            $(".leaflet-interactive").remove();
            $(".leaflet-popup").remove();
            $.ajax({
                url: liga,
                type: "GET",
                dateType: "json",
            }).done(function(data){
                $.each(data, function(key, datos){
                    //Obtiene las corrdenadas en formato geoJSON
                    coordenadaCalles = datos.location.value
                    //Obtiene la ocupación para la condición del color de la linea
                    ocupacion = datos.occupancy.value
                    //console.log(datos)
                    if(ocupacion <= 0.3){
                    var myLayer = L.geoJSON(coordenadaCalles, {style: bajo}).addTo(map);
                    }else if(ocupacion >0.3 && ocupacion <= 0.7){
                        var myLayer = L.geoJSON(coordenadaCalles, {style: medio}).addTo(map);
                    }else{
                        var myLayer = L.geoJSON(coordenadaCalles, {style: alto}).addTo(map);
                    }

                    idCalle = datos.id
                    direccion = datos.address.value.streetAddress
                    localidad = datos.address.value.addressLocality
                    pais = datos.address.value.addressCountry
                    velocidad = datos.averageVehicleSpeed.value
                    horaObservacion = datos.dateObserved.value
                    porcentajeOcupacion = ocupacion*100

                    //console.log(velocidad)
                    myLayer.bindPopup('<h3>Información de Trafico</h3><p>ID: '+ idCalle
+'<br>Dirección: '+ direccion +', '+ localidad +', '+ pais +'<br>Velocidad: ' + velocidad + '
Km/h<br>Porcentaje Ocupación: '+ porcentajeOcupacion.toFixed(1) +'%<br>hora: ' +
horaObservacion);
                })
```

```
            })
        }

        //Pinta la ocupación de las calles, avenidas, bulevares en el mapa, se generan 5 ligas debido a que FIWARA se puede copnsumir entidades de 1000 en 1000 -- 240000 = a 4min
        setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=0")',60000); //1000
        setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=1000")',60000); //1000
        setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=2000")',60000);//1000
        setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=3000")',60000); //1000
        setInterval('pintaOcupacion("http://172.16.10.220:1026/v2/entities?type=TrafficFlowObserved&idPattern=^TrafficFlowObserved:CDMX&limit=1000&offset=4000")',60000) //132
    </script>
</body>
</html>
```

Mapa que representa el comportamiento de un Metrobús específico durante un lapso de tiempo específico.
mapaMetrobusHistoricos.php

```
<!DOCTYPE html>
<html>
<head><title>Metrobuses Prueba</title>
<h3>Mapa Metrobuses CDMX</h3>
    <meta charset="UTF-8" />

    <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"integrity="sha512-XQoYMqMTK8LvdxXYG3nZ448hOEQiglfqkJs1NOQV44cWnUrBc8PkAOcXy20w0vlaXaVUearIOBhiXZ5V3ynxwA==" crossorigin=""></script>

    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" integrity="sha512-xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZMZ19scR4PsZChSR7A==" crossorigin=""/>

    <script src="https://code.jquery.com/jquery-3.5.1.js"></script>

</head>
<body>
    <!-- div para el tamaño del mapa -->
    <div id="map" style="width: 1900px; height: 850px"></div>
```

```
<!-- script para dibujar el mapa -->
<button>Busca Metrobús</button>

<script>
    //var map=L.map('map',{[41.66,-4.72],zoom:10});
    var map = L.map('map').setView([19.427591, -99.132965],15);
    var layer=new L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png');
    map.addLayer(layer);
</script>

<!-- Script para dibujar calles -->
<script>
    //Propiedades de la linea que se va a pintar
    var customIcon = new L.Icon({
        iconUrl: 'bus_icon3.png',
        iconSize: [30, 30],
        iconAnchor: [20, 0]
    });
    //Consulta de coordenadas y dibujo de calles
    function consutalDatos(liga){
        return $.ajax({
            url: liga,
            type: "GET",
            dateType: "json",
        }).done(function(data){
            //console.log(data.attributes[0].values[0])
            //return JSON.parse('{"name":"John", "age":30, "city":"New York"}');
        })
    }

    function sleep(milliseconds) {
        return new Promise(resolve => setTimeout(resolve, milliseconds));
    }

    var customOptions = {
    'maxWidth': '500',
    'className' : 'custom',

    }

    const button = document.querySelector('button');

    button.onclick = function() {
        let noMetrobus = prompt('Número de Unidad');
```

```
            let fechaInicio = prompt('Fecha y hora de inicio (YYYY-MM-DDTHH:MM:SS) Ej. 2022-10-
20T00:00:00'); //2022-09-10T00:00:00  empieza desde aquí
            let fechaFin = prompt('Fecha y hora final (YYYY-MM-DDTHH:MM:SS) Ej. Ej. 2022-10-
20T23:59:00');

            var idMetrobus = "Vehicle:MetrobusCDMX:"+noMetrobus;

            //Con fines de prueba solo estan descargando 10 datos, para la prueba completa quitar
el +"&limit=10"
            consutalDatos("http://172.16.10.220:8668/v2/entities/"+idMetrobus+"?fromDate="+fechaI
nicio+"&toDate="+fechaFin).done(function(data){
                //console.log(data)
                tamanoArreglo = data.attributes[1].values.length
                async function pintaMetrobus(){
                    for (var i = 0; i < tamanoArreglo; i++) {
                        await sleep(1000);
                        $(".leaflet-marker-icon").remove();
                        $(".leaflet-popup").remove();
                        var heading = data.attributes[2].values[i]
                        var latitud = data.attributes[3].values[i].coordinates[1]
                        var longitud =  data.attributes[3].values[i].coordinates[0]
                        var odometro = data.attributes[5].values[i]
                        var fecha = data.attributes[0].values[i]
                        var estatus = data.attributes[6].values[i]
                        var velocidad = data.attributes[7].values[i]
                        //console.log(longitud)

                        var marker = L.marker([latitud, longitud], {icon: customIcon});
                        marker.addTo(map);

                        marker.bindPopup("<h4>Id Metrobús: "+ idMetrobus  +"</h1> <p>ServiceType:
Public<br>Heading: "+ heading +"<br> Latitude: "+ latitud +"<br> Longitude:"+ longitud +"
<br>Odometer: "+ odometro +"<br> ServiceStatus: "+ estatus +"<br> Speed(KM/H): "+ velocidad
+"<br> Time: "+ fecha +"</p>", customOptions).openPopup();

                        document.getElementById('infoDatos').innerHTML="Dato numero "+ (i+1) +"
de "+tamanoArreglo;
                    }
                }
                pintaMetrobus()
            })
        }
    </script>
    <div id="infoDatos"></div>
</body>
```

```
</html>
```

Mapa que representa el comportamiento de la congestión vehicular de un segmento de la Ciudad de México en un horario especifico.
mapaTraficoOcupacionHistorico.php

```
<!DOCTYPE html>
<html>
<head><title>Trafico Prueba Historicos</title>
<h3>Mapa Trafico Pruebas Historicos</h3>
    <meta charset="UTF-8" />

    <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"integrity="sha512-
XQoYMqMTK8LvdxXYG3nZ448hOEQiglfqkJs1NOQV44cWnUrBc8PkAOcXy20w0vlaXaVUearIOBhiXZ5V3ynxwA=="crossori
gin=""></script>

    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
integrity="sha512-
xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZMZ19scR4PsZChSR7A=="crossori
gin=""/>

    <script src="https://code.jquery.com/jquery-3.5.1.js"></script>

</head>
<body>
    <!-- div para el tamaño del mapa -->
    <div id="map" style="width: 1900px; height: 850px"></div>

    <button>Busca Segmento</button>

    <!-- script para dibujar el mapa -->
    <script>
        //var map=L.map('map',{[41.66,-4.72],zoom:10});
        var map = L.map('map').setView([19.427591, -99.132965],30);
        var layer=new L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png');
        map.addLayer(layer);
    </script>

    <!-- Script para dibujar calles -->
    <script>
        //Propiedades de la linea que se va a pintar
        var alto = {
            "color": "#8a030a",
            "weight": 10,
            "opacity": 0.70
        };
```

```
        var medio = {
            "color": "#b9770e",
            "weight": 10,
            "opacity": 0.70
        };
        var bajo = {
            "color": "#04b316",
            "weight": 10,
            "opacity": 0.70
        };

        //Consulta de coordenadas y dibujo de calles
        function consutalDatos(liga){
            return $.ajax({
                url: liga,
                type: "GET",
                dateType: "json",
            }).done(function(data){
                //console.log(data.attributes[0].values[0])
            })
        }

        function sleep(milliseconds) {
            return new Promise(resolve => setTimeout(resolve, milliseconds));
        }

        function datosHistoricosTrafico(idSegmento, fechaInicio, fechaFinal){
            consutalDatos("http://172.16.10.220:1026/v2/entities/"+idSegmento).done(function(data
){
                coordenadaCalles = data.location.value
                direccion = data.address.value.streetAddress
                localidad = data.address.value.addressLocality
                pais = data.address.value.addressCountry
                consutalDatos("http://172.16.10.220:8668/v2/entities/"+idSegmento+"?fromDate="+fe
chaInicio+"&toDate="+fechaFinal).done(function(data2){
                    //console.log(data2)
                    tamanoArreglo = data2.attributes[1].values.length
                    async function pintaCalle(){
                        //do{
                            for (var i = 0; i < tamanoArreglo; i++) {
                                await sleep(1000);
                                $(".leaflet-interactive").remove();
                                $(".leaflet-popup").remove();
                                var fecha = data2.attributes[1].values[i]
                                var velocidad = data2.attributes[0].values[i]
```

```
                                var ocupacion = data2.attributes[3].values[i]
                                //console.log("id: " + idSegmento)
                                //console.log("Fecha: "+fecha)
                                //console.log("velocudad: "+velocidad)
                                //console.log("ocupación: "+ocupacion)

                                if(ocupacion <= 0.3){
                                var myLayer = L.geoJSON(coordenadaCalles, {style:
bajo}).addTo(map);
                                }else if(ocupacion >0.3 && ocupacion <= 0.7){
                                    var myLayer = L.geoJSON(coordenadaCalles, {style:
medio}).addTo(map);
                                }else{
                                    var myLayer = L.geoJSON(coordenadaCalles, {style:
alto}).addTo(map);
                                }
                                porcentajeOcupacion = ocupacion*100
                                myLayer.bindPopup('<h3>Información de Trafico</h3><p>ID: '+
idSegmento +'<br>Dirección: '+ direccion +', '+ localidad +', '+ pais +'<br>Velocidad: ' +
velocidad + ' Km/h<br>Porcentaje Ocupación: '+ porcentajeOcupacion.toFixed(1) +'%<br>hora: ' +
fecha).openPopup();

                                document.getElementById('infoDatos').innerHTML="Dato numero "+
(i+1) +" de "+tamanoArreglo;
                            }
                    }
                    pintaCalle()
                })
            })
        }

        const button = document.querySelector('button');

        button.onclick = function() {
            let idSegmento = prompt('ID del Segmento');
            let fechaInicio = prompt('Fecha y hora de inicio (YYYY-MM-DDTHH:MM:SS) Ej. 2022-10-
22T00:00:00'); //2022-09-10T00:00:00  empieza desde aquí
            let fechaFinal = prompt('Fecha y hora final (YYYY-MM-DDTHH:MM:SS) Ej. 2022-10-
22T23:59:59');
            datosHistoricosTrafico(idSegmento, fechaInicio, fechaFinal)
        }

    </script>

    <div id="infoDatos"></div>
```

```
</body>
</html>
```

Anexo C

Resultados del análisis de los datos del espacio unificado

C.1 Resultados obtenidos para el monitoreo de los datos de los Metrobuses de la CDMX

En las tablas Tabla C1 a la Tabla C30 se presentan los resultados obtenidos al monitorear los datos de los Metrobuses de la CDMX. Se utilizan 30 unidades elegidas al azar.

Tabla C1 Análisis de datos del Metrobús 1021

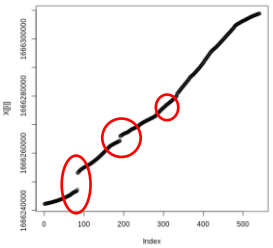
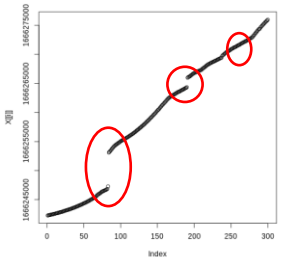
<table>
<tr><td colspan="4">Vehicle:MetrobúsCDMX:1021</td></tr>
<tr><td>2022-10-20 05:04:32
Fecha y hora de inicio de monitoreo</td><td>2022-10-20 23:33:56
Fecha y hora final del monitoreo</td><td>66564
Periodo de funcionamiento del Metrobús (segundos)</td><td>16035
Total de tiempo no reportado (segundos)</td></tr>
<tr><td>123
Tiempo promedio de actualización (segundos)</td><td></td><td></td><td></td></tr>
<tr><td colspan="2"> 
Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.</td><td colspan="2"> 
Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)</td></tr>
<tr><td>18.49
Periodo de funcionamiento del Metrobús (horas)</td><td>2.05
Tiempo promedio de actualización (minutos)</td><td>4.45
Tiempo no reportado (horas)</td><td>24
Porcentaje de tiempo no reportado</td></tr>
</table>

Tabla C2 Análisis de datos del Metrobús 1302

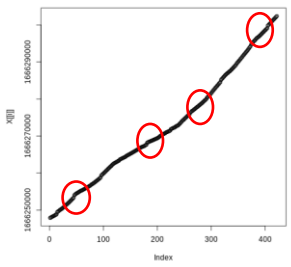
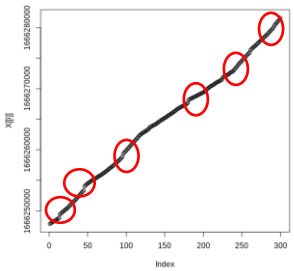
<table>
<tr><td colspan="4">Vehicle:MetrobúsCDMX:1302</td></tr>
<tr><td>2022-10-20 06:38:06
Fecha y hora de inicio de monitoreo</td><td>2022-10-20 21:46:32
Fecha y hora final del monitoreo</td><td>54506
Periodo de funcionamiento del Metrobús (segundos)</td><td>9510
Total de tiempo no reportado (segundos)</td></tr>
<tr><td>129
Tiempo promedio de actualización (segundos)</td><td></td><td></td><td></td></tr>
<tr><td colspan="2"> 
Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.</td><td colspan="2"> 
Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)</td></tr>
<tr><td>15.14
Periodo de funcionamiento del Metrobús (horas)</td><td>2.15
Tiempo promedio de actualización (minutos)</td><td>2.64
Tiempo no reportado (horas)</td><td>17
Porcentaje de tiempo no reportado</td></tr>
</table>

Tabla C3 Análisis de datos del Metrobús 607

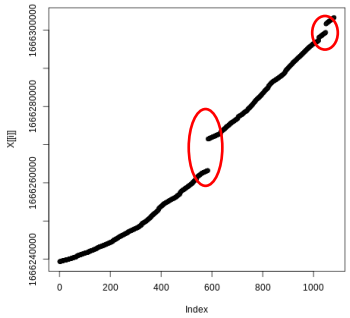
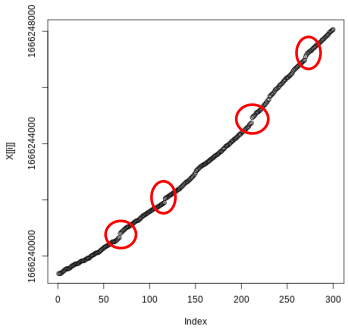
Vehicle:MetrobúsCDMX:607			
2022-10-20 04:16:20 Fecha y hora de inicio de monitoreo	2022-10-20 22:01:54 Fecha y hora final del monitoreo	63934 Periodo de funcionamiento del Metrobús (segundos)	20540 Total de tiempo no reportado (segundos)
59 Tiempo promedio de actualización (segundos)			
			
Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
17.76 Periodo de funcionamiento del Metrobús (horas)	0.98 Tiempo promedio de actualización (minutos)	5.7 Tiempo no reportado (horas)	32 Porcentaje de tiempo no reportado

Tabla C4 Análisis de datos del Metrobús 9402

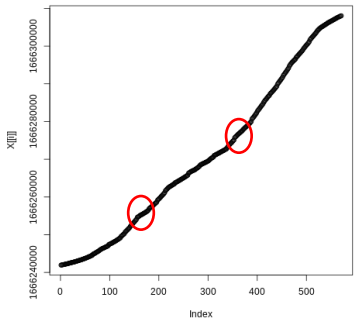
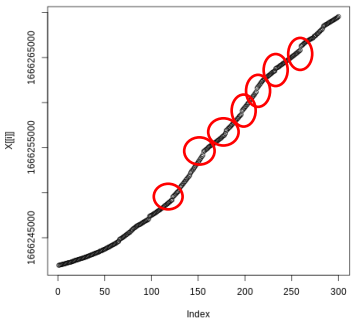
Vehicle:MetrobúsCDMX:9402			
2022-10-20 04:59:02 Fecha y hora de inicio de monitoreo	2022-10-20 23:20:50 Fecha y hora final del monitoreo	66108 Periodo de funcionamiento del Metrobús (segundos)	14641 Total de tiempo no reportado (segundos)
116 Tiempo promedio de actualización (segundos)			
			
Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
18.36 Periodo de funcionamiento del Metrobús (horas)	1.93 Tiempo promedio de actualización (minutos)	4 Tiempo no reportado (horas)	22 Porcentaje de tiempo no reportado

Tabla C5 Análisis de datos del Metrobús 807

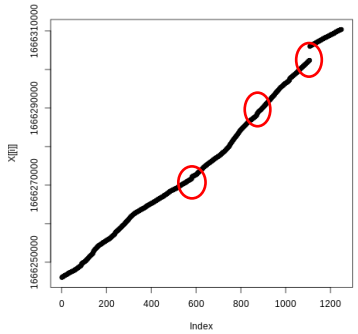
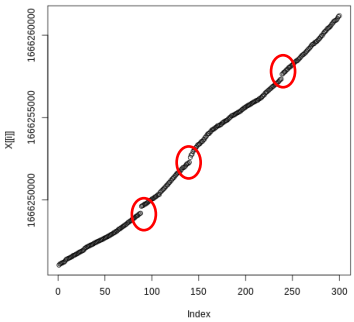
Vehicle:MetrobúsCDMX:807			
2022-10-20 06:07:20 Fecha y hora de inicio de monitoreo	2022-10-20 23:59:57 Fecha y hora final del monitoreo	64357 Periodo de funcionamiento del Metrobús (segundos)	16381 Total de tiempo no reportado (segundos)
52 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
17.88 Periodo de funcionamiento del Metrobús (horas)	0.86 Tiempo promedio de actualización (minutos)	4.55 Tiempo no reportado (horas)	25 Porcentaje de tiempo no reportado

Tabla C6 Análisis de datos del Metrobús 229

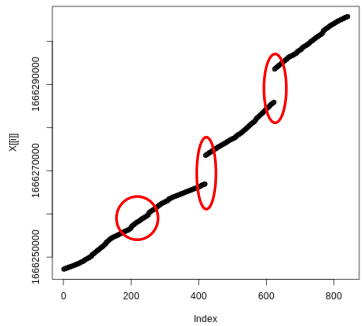
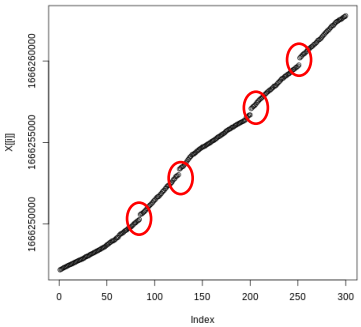
Vehicle:MetrobúsCDMX:229			
2022-10-20 06:26:28 Fecha y hora de inicio de monitoreo	2022-10-20 22:42:04 Fecha y hora final del monitoreo	58536 Periodo de funcionamiento del Metrobús (segundos)	19613.13 Total de tiempo no reportado (segundos)
69.6 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
16.26 Periodo de funcionamiento del Metrobús (horas)	1.16 Tiempo promedio de actualización (minutos)	5.45 Tiempo no reportado (horas)	34 Porcentaje de tiempo no reportado

Tabla C7 Análisis de datos del Metrobús 1112

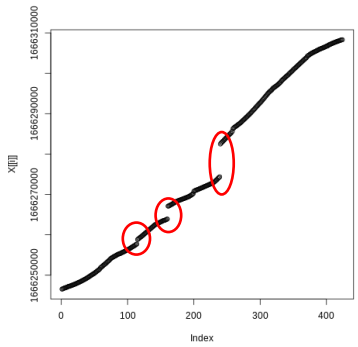
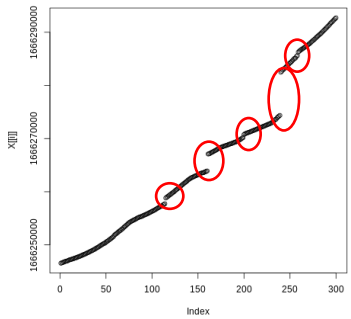
Vehicle:MetrobúsCDMX:1112			
2022-10-20 06:15:54 Fecha y hora de inicio de monitoreo	2022-10-20 23:25:10 Fecha y hora final del monitoreo	61756 Periodo de funcionamiento del Metrobús (segundos)	16535.28 Total de tiempo no reportado (segundos)
145.65 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
17.15 Periodo de funcionamiento del Metrobús (horas)	2.43 Tiempo promedio de actualización (minutos)	4.59 Tiempo no reportado (horas)	27 Porcentaje de tiempo no reportado

Tabla C8 Análisis de datos del Metrobús 628

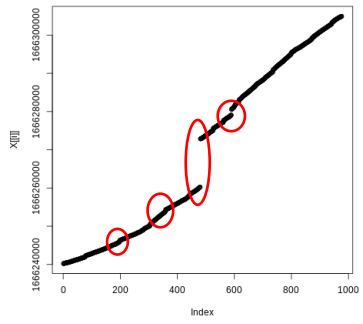
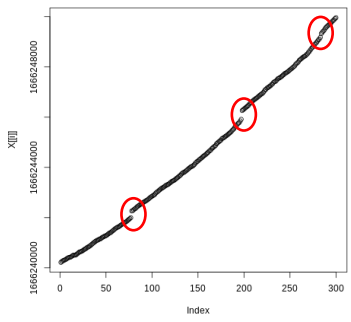
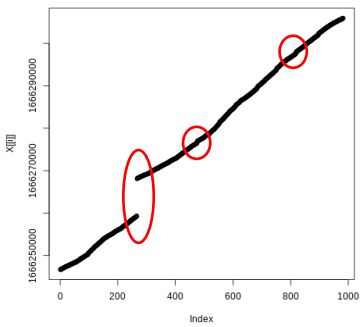
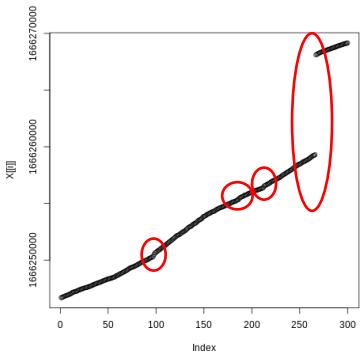
Vehicle:MetrobúsCDMX:628			
2022-10-20 04:30:16 Fecha y hora de inicio de monitoreo	2022-10-20 22:28:32 Fecha y hora final del monitoreo	64696 Periodo de funcionamiento del Metrobús (segundos)	22410.89 Total de tiempo no reportado (segundos)
66.29 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
17.97 Periodo de funcionamiento del Metrobús (horas)	1.1 Tiempo promedio de actualización (minutos)	6.23 Tiempo no reportado (horas)	35 Porcentaje de tiempo no reportado

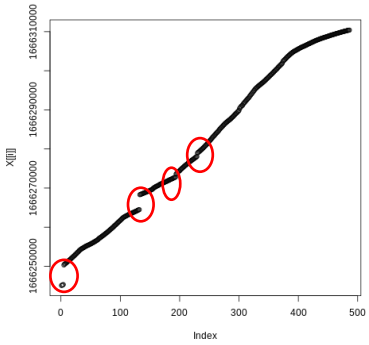
Tabla C9 Análisis de datos del Metrobús 774

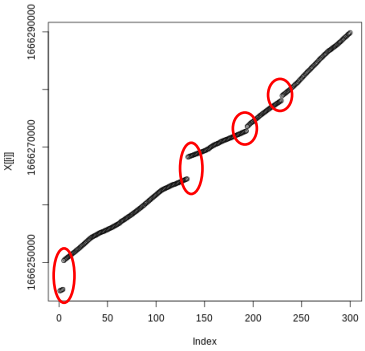
<table>
<tr><td colspan="4">Vehicle:MetrobúsCDMX:774</td></tr>
<tr><td>2022-10-20 06:18:04
Fecha y hora de inicio de monitoreo</td><td>2022-10-20 22:45:45
Fecha y hora final del monitoreo</td><td>59261
Periodo de funcionamiento del Metrobús (segundos)</td><td>16409.85
Total de tiempo no reportado (segundos)</td></tr>
<tr><td>60.35
Tiempo promedio de actualización (segundos)</td><td></td><td></td><td></td></tr>
<tr><td colspan="2"> 

Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.</td><td colspan="2"> 

Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)</td></tr>
<tr><td>16.46
Periodo de funcionamiento del Metrobús (horas)</td><td>1.01
Tiempo promedio de actualización (minutos)</td><td>4.56
Tiempo no reportado (horas)</td><td>28
Porcentaje de tiempo no reportado</td></tr>
</table>

Tabla C10 Análisis de datos del Metrobús 1114

<table>
<tr><td colspan="4">Vehicle:MetrobúsCDMX:1114</td></tr>
<tr><td>2022-10-20 05:51:22
Fecha y hora de inicio de monitoreo</td><td>2022-10-20 23:59:58
Fecha y hora final del monitoreo</td><td>65316
Periodo de funcionamiento del Metrobús (segundos)</td><td>15696.1
Total de tiempo no reportado (segundos)</td></tr>
<tr><td>134.4
Tiempo promedio de actualización (segundos)</td><td></td><td></td><td></td></tr>
<tr><td colspan="2"> 

Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.</td><td colspan="2"> 

Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)</td></tr>
<tr><td>18.14
Periodo de funcionamiento del Metrobús (horas)</td><td>2.24
Tiempo promedio de actualización (minutos)</td><td>4.36
Tiempo no reportado (horas)</td><td>24
Porcentaje de tiempo no reportado</td></tr>
</table>

Tabla C11 Análisis de datos del Metrobús 2607

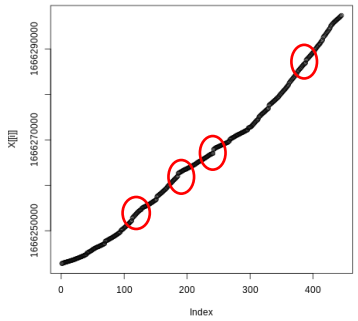
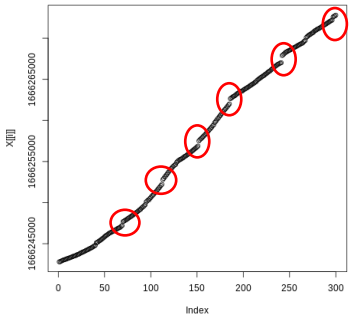
Vehicle:MetrobúsCDMX:2607			
2022-10-20 05:13:15 Fecha y hora de inicio de monitoreo	2022-10-20 20:23:14 Fecha y hora final del monitoreo	54599 Periodo de funcionamiento del Metrobús (segundos)	11994.01 Total de tiempo no reportado (segundos)
122.69 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
15.17 Periodo de funcionamiento del Metrobús (horas)	2.04 Tiempo promedio de actualización (minutos)	3.33 Tiempo no reportado (horas)	22 Porcentaje de tiempo no reportado

Tabla C12 Análisis de datos del Metrobús 791

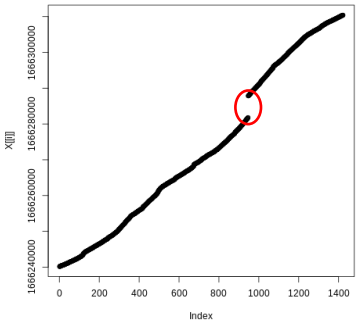
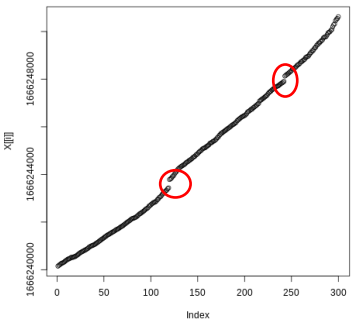
Vehicle:MetrobúsCDMX:791			
2022-10-20 04:29:11 Fecha y hora de inicio de monitoreo	2022-10-20 23:59:54 Fecha y hora final del monitoreo	70243 Periodo de funcionamiento del Metrobús (segundos)	18790.09 Total de tiempo no reportado (segundos)
49.4 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
19.51 Periodo de funcionamiento del Metrobús (horas)	0.82 Tiempo promedio de actualización (minutos)	5.22 Tiempo no reportado (horas)	27 Porcentaje de tiempo no reportado

Tabla C13 Análisis de datos del Metrobús 554

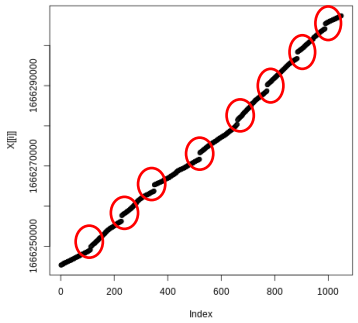
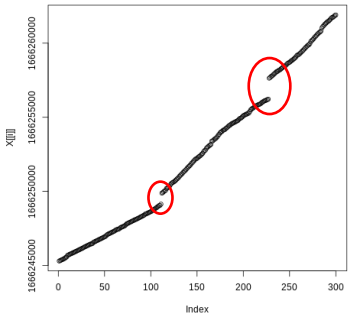
Vehicle:MetrobúsCDMX:554			
2022-10-20 05:55:01 Fecha y hora de inicio de monitoreo	2022-10-20 23:10:14 Fecha y hora final del monitoreo	62113 Periodo de funcionamiento del Metrobús (segundos)	17180.19 Total de tiempo no reportado (segundos)
59.21 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
17.25 Periodo de funcionamiento del Metrobús (horas)	0.99 Tiempo promedio de actualización (minutos)	4.77 Tiempo no reportado (horas)	28 Porcentaje de tiempo no reportado

Tabla C14 Análisis de datos del Metrobús 602

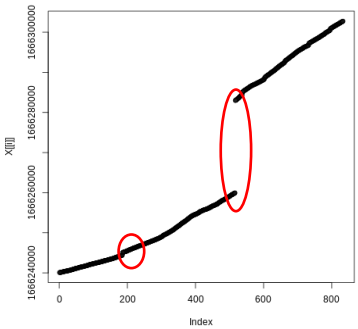
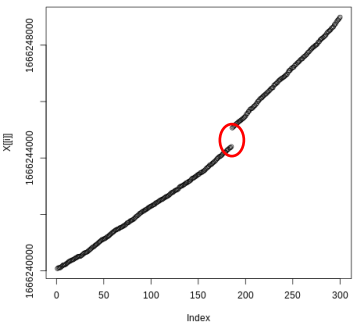
Vehicle:MetrobúsCDMX:602			
2022-10-20 04:28:02 Fecha y hora de inicio de monitoreo	2022-10-20 21:52:33 Fecha y hora final del monitoreo	62671 Periodo de funcionamiento del Metrobús (segundos)	27277.82 Total de tiempo no reportado (segundos)
75.24 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
17.41 Periodo de funcionamiento del Metrobús (horas)	1.25 Tiempo promedio de actualización (minutos)	7.58 Tiempo no reportado (horas)	44 Porcentaje de tiempo no reportado

Tabla C15 Análisis de datos del Metrobús 822

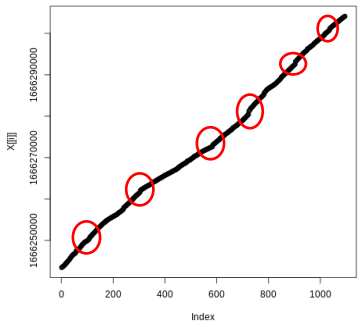
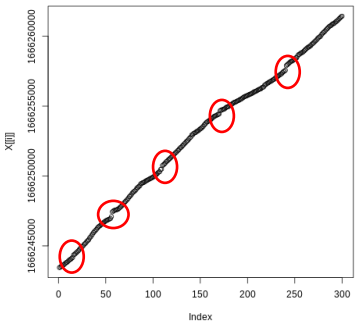
Vehicle:MetrobúsCDMX:822			
2022-10-20 05:24:08 Fecha y hora de inicio de monitoreo	2022-10-20 22:15:52 Fecha y hora final del monitoreo	60704 Periodo de funcionamiento del Metrobús (segundos)	14104.59 Total de tiempo no reportado (segundos)
55.44 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
16.86 Periodo de funcionamiento del Metrobús (horas)	0.92 Tiempo promedio de actualización (minutos)	3.92 Tiempo no reportado (horas)	23 Porcentaje de tiempo no reportado

Tabla C16 Análisis de datos del Metrobús 1310

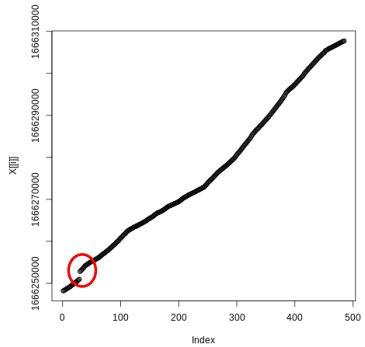
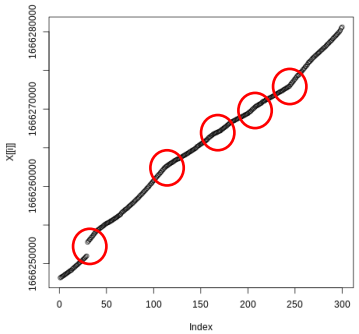
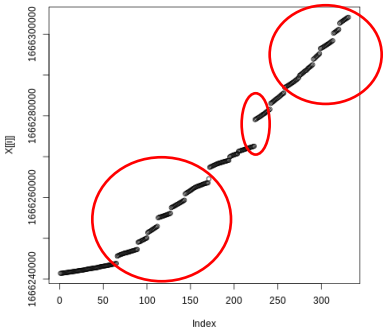
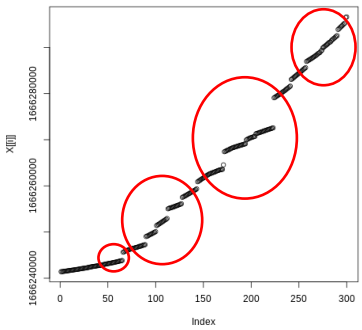
Vehicle:MetrobúsCDMX:1310			
2022-10-20 06:43:02 Fecha y hora de inicio de monitoreo	2022-10-20 23:15:44 Fecha y hora final del monitoreo	59562 Periodo de funcionamiento del Metrobús (segundos)	10405.19 Total de tiempo no reportado (segundos)
122.81 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
16.55 Periodo de funcionamiento del Metrobús (horas)	2.05 Tiempo promedio de actualización (minutos)	2.89 Tiempo no reportado (horas)	17 Porcentaje de tiempo no reportado

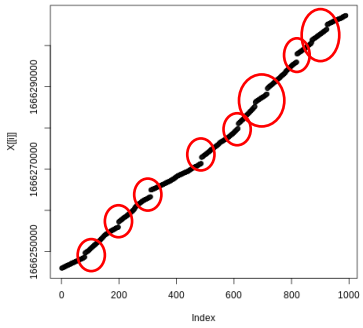
Tabla C17 Análisis de datos del Metrobús 2515

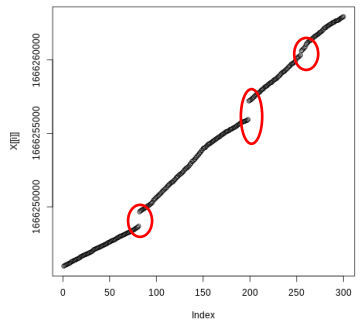
<table>
<tr><td colspan="4">Vehicle:MetrobúsCDMX:2515</td></tr>
<tr><td>2022-10-20 04:50:21
Fecha y hora de inicio de monitoreo</td><td>2022-10-20 22:16:01
Fecha y hora final del monitoreo</td><td>62740
Periodo de funcionamiento del Metrobús (segundos)</td><td>27699.05
Total de tiempo no reportado (segundos)</td></tr>
<tr><td>189.55
Tiempo promedio de actualización (segundos)</td><td></td><td></td><td></td></tr>
<tr><td colspan="2"> 

Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.</td><td colspan="2"> 

Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)</td></tr>
<tr><td>17.43
Periodo de funcionamiento del Metrobús (horas)</td><td>3.16
Tiempo promedio de actualización (minutos)</td><td>7.69
Tiempo no reportado (horas)</td><td>44
Porcentaje de tiempo no reportado</td></tr>
</table>

Tabla C18 Análisis de datos del Metrobús 568

<table>
<tr><td colspan="4">Vehicle:MetrobúsCDMX:568</td></tr>
<tr><td>2022-10-20 06:06:05
Fecha y hora de inicio de monitoreo</td><td>2022-10-20 23:08:06
Fecha y hora final del monitoreo</td><td>61321
Periodo de funcionamiento del Metrobús (segundos)</td><td>17263.38
Total de tiempo no reportado (segundos)</td></tr>
<tr><td>62
Tiempo promedio de actualización (segundos)</td><td></td><td></td><td></td></tr>
<tr><td colspan="2"> 

Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.</td><td colspan="2"> 

Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)</td></tr>
<tr><td>17.03
Periodo de funcionamiento del Metrobús (horas)</td><td>1.03
Tiempo promedio de actualización (minutos)</td><td>4.8
Tiempo no reportado (horas)</td><td>28
Porcentaje de tiempo no reportado</td></tr>
</table>

Tabla C19 Análisis de datos del Metrobús 766

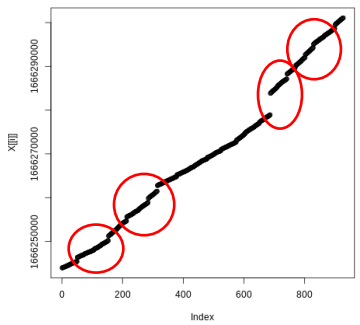
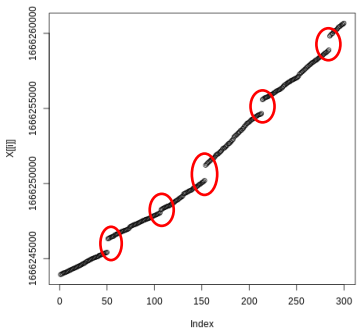
Vehicle:MetrobúsCDMX:766			
2022-10-20 05:32:17 Fecha y hora de inicio de monitoreo	2022-10-20 21:24:38 Fecha y hora final del monitoreo	57141 Periodo de funcionamiento del Metrobús (segundos)	18455.61 Total de tiempo no reportado (segundos)
61.64 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
15.87 Periodo de funcionamiento del Metrobús (horas)	1.03 Tiempo promedio de actualización (minutos)	5.13 Tiempo no reportado (horas)	32 Porcentaje de tiempo no reportado

Tabla C20 Análisis de datos del Metrobús 815

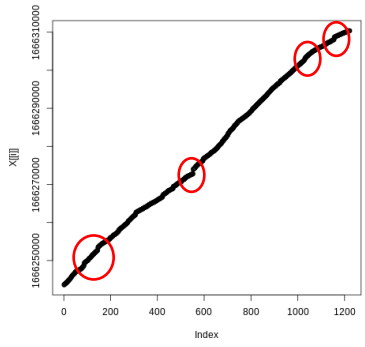
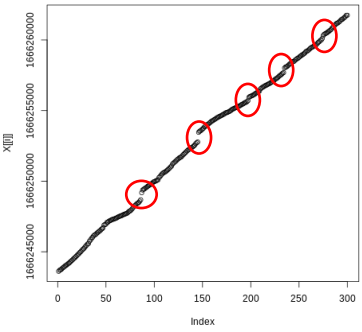
Vehicle:MetrobúsCDMX:815			
2022-10-20 05:26:56 Fecha y hora de inicio de monitoreo	2022-10-20 23:59:32 Fecha y hora final del monitoreo	66756 Periodo de funcionamiento del Metrobús (segundos)	16782.39 Total de tiempo no reportado (segundos)
54.63 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
18.54 Periodo de funcionamiento del Metrobús (horas)	0.91 Tiempo promedio de actualización (minutos)	4.66 Tiempo no reportado (horas)	25 Porcentaje de tiempo no reportado

Tabla C21 Análisis de datos del Metrobús 904

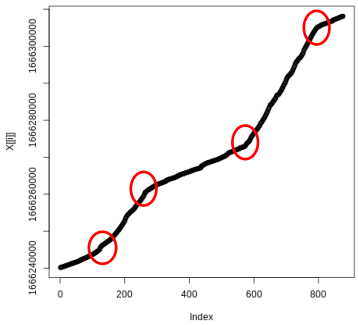
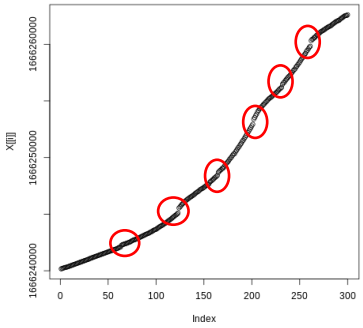
Vehicle:MetrobúsCDMX:904			
2022-10-20 04:29:28 Fecha y hora de inicio de monitoreo	2022-10-20 23:22:12 Fecha y hora final del monitoreo	67964 Periodo de funcionamiento del Metrobús (segundos)	22116.06 Total de tiempo no reportado (segundos)
77.58 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
18.88 Periodo de funcionamiento del Metrobús (horas)	1.29 Tiempo promedio de actualización (minutos)	6.14 Tiempo no reportado (horas)	33 Porcentaje de tiempo no reportado

Tabla C22 Análisis de datos del Metrobús 768

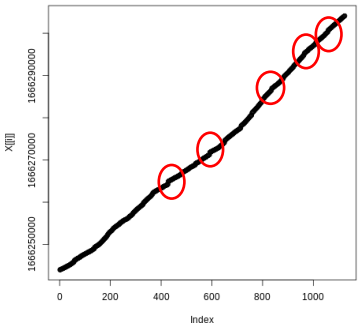
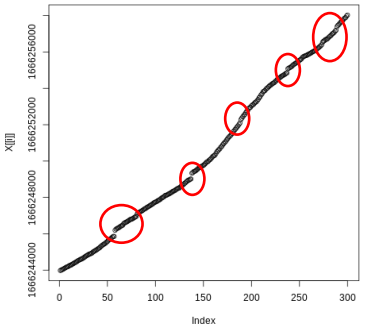
Vehicle:MetrobúsCDMX:768			
2022-10-20 05:33:00 Fecha y hora de inicio de monitoreo	2022-10-20 22:15:44 Fecha y hora final del monitoreo	60164 Periodo de funcionamiento del Metrobús (segundos)	14879.16 Total de tiempo no reportado (segundos)
53.53 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
16.71 Periodo de funcionamiento del Metrobús (horas)	0.89 Tiempo promedio de actualización (minutos)	4.13 Tiempo no reportado (horas)	25 Porcentaje de tiempo no reportado

Tabla C23 Análisis de datos del Metrobús 727

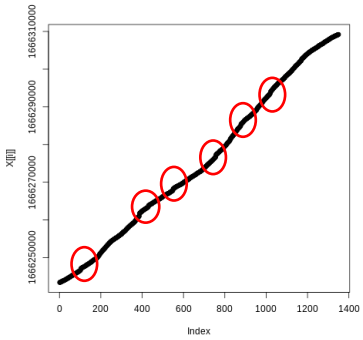
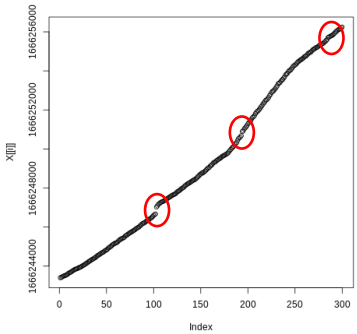
Vehicle:MetrobúsCDMX:727			
2022-10-20 05:23:23 Fecha y hora de inicio de monitoreo	2022-10-20 23:40:11 Fecha y hora final del monitoreo	65808 Periodo de funcionamiento del Metrobús (segundos)	15597.99 Total de tiempo no reportado (segundos)
48.75 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
18.28 Periodo de funcionamiento del Metrobús (horas)	0.81 Tiempo promedio de actualización (minutos)	4.33 Tiempo no reportado (horas)	24 Porcentaje de tiempo no reportado

Tabla C24 Análisis de datos del Metrobús 470

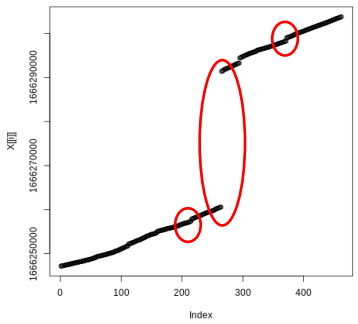
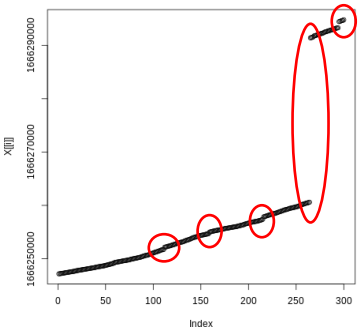
Vehicle:MetrobúsCDMX:470			
2022-10-20 06:26:07 Fecha y hora de inicio de monitoreo	2022-10-20 22:10:15 Fecha y hora final del monitoreo	56648 Periodo de funcionamiento del Metrobús (segundos)	33338.24 Total de tiempo no reportado (segundos)
122.61 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
15.74 Periodo de funcionamiento del Metrobús (horas)	2.04 Tiempo promedio de actualización (minutos)	9.26 Tiempo no reportado (horas)	59 Porcentaje de tiempo no reportado

Tabla C25 Análisis de datos del Metrobús 398

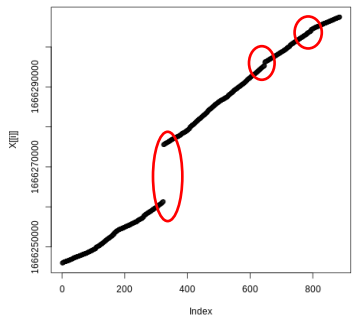
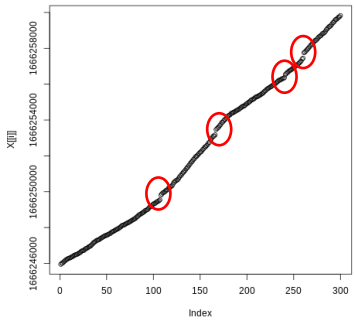
Vehicle:MetrobúsCDMX:398			
2022-10-20 06:05:49 Fecha y hora de inicio de monitoreo	2022-10-20 23:11:27 Fecha y hora final del monitoreo	61538 Periodo de funcionamiento del Metrobús (segundos)	20262.01 Total de tiempo no reportado (segundos)
69.46 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
17.09 Periodo de funcionamiento del Metrobús (horas)	1.16 Tiempo promedio de actualización (minutos)	5.63 Tiempo no reportado (horas)	33 Porcentaje de tiempo no reportado

Tabla C26 Análisis de datos del Metrobús 3346

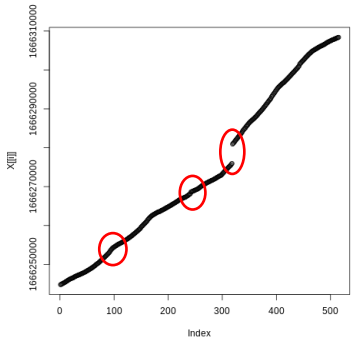
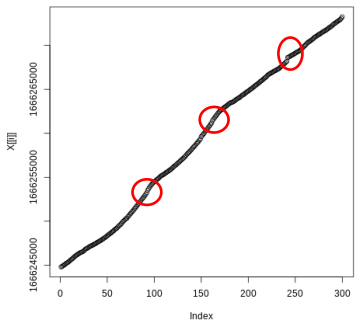
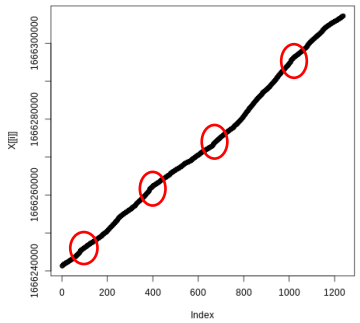
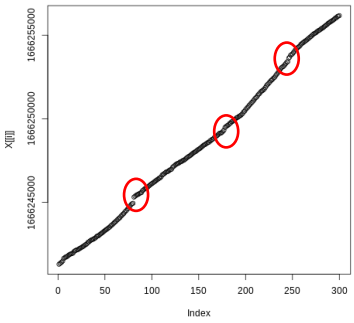
Vehicle:MetrobúsCDMX:3346			
2022-10-20 05:46:39 Fecha y hora de inicio de monitoreo	2022-10-20 23:26:37 Fecha y hora final del monitoreo	63598 Periodo de funcionamiento del Metrobús (segundos)	13552.24 Total de tiempo no reportado (segundos)
123.49 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
17.67 Periodo de funcionamiento del Metrobús (horas)	2.06 Tiempo promedio de actualización (minutos)	3.76 Tiempo no reportado (horas)	21 Porcentaje de tiempo no reportado

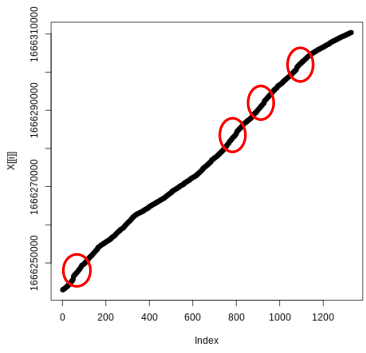
Tabla C27 Análisis de datos del Metrobús 800

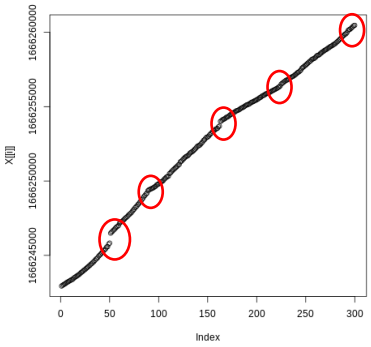
<table>
<tr><td colspan="4">Vehicle:MetrobúsCDMX:800</td></tr>
<tr><td>2022-10-20 04:48:22
Fecha y hora de inicio de monitoreo</td><td>2022-10-20 23:07:23
Fecha y hora final del monitoreo</td><td>65941
Periodo de funcionamiento del Metrobús (segundos)</td><td>13572.74
Total de tiempo no reportado (segundos)</td></tr>
<tr><td>53.26
Tiempo promedio de actualización (segundos)</td><td></td><td></td><td></td></tr>
<tr><td colspan="2"> 

Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.</td><td colspan="2"> 

Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)</td></tr>
<tr><td>18.32
Periodo de funcionamiento del Metrobús (horas)</td><td>0.89
Tiempo promedio de actualización (minutos)</td><td>3.77
Tiempo no reportado (horas)</td><td>21
Porcentaje de tiempo no reportado</td></tr>
</table>

Tabla C28 Análisis de datos del Metrobús 827

<table>
<tr><td colspan="4">Vehicle:MetrobúsCDMX:827</td></tr>
<tr><td>2022-10-20 05:15:38
Fecha y hora de inicio de monitoreo</td><td>2022-10-20 23:59:44
Fecha y hora final del monitoreo</td><td>67446
Periodo de funcionamiento del Metrobús (segundos)</td><td>15477
Total de tiempo no reportado (segundos)</td></tr>
<tr><td>50.75
Tiempo promedio de actualización (segundos)</td><td></td><td></td><td></td></tr>
<tr><td colspan="2"> 

Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.</td><td colspan="2"> 

Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)</td></tr>
<tr><td>18.74
Periodo de funcionamiento del Metrobús (horas)</td><td>0.85
Tiempo promedio de actualización (minutos)</td><td>4.3
Tiempo no reportado (horas)</td><td>23
Porcentaje de tiempo no reportado</td></tr>
</table>

Tabla C29 Análisis de datos del Metrobús 845

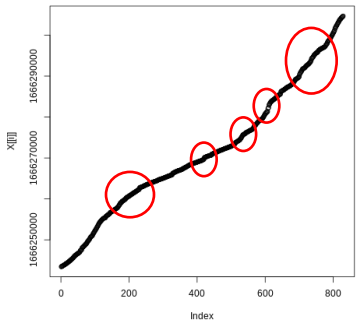
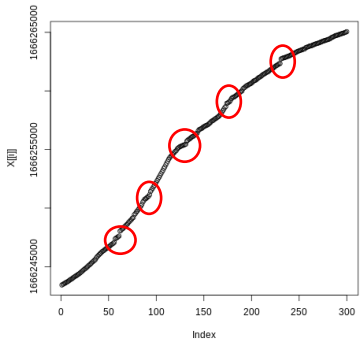
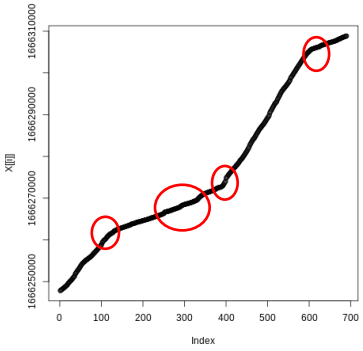
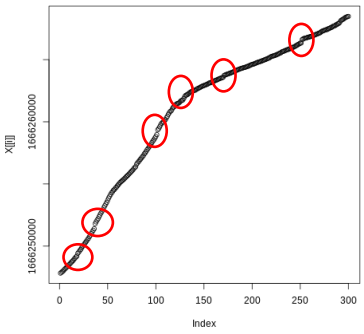
Vehicle:MetrobúsCDMX:845			
2022-10-20 05:23:53 Fecha y hora de inicio de monitoreo	2022-10-20 22:24:29 Fecha y hora final del monitoreo	61236 Periodo de funcionamiento del Metrobús (segundos)	18353 Total de tiempo no reportado (segundos)
73.87 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
17.01 Periodo de funcionamiento del Metrobús (horas)	1.23 Tiempo promedio de actualización (minutos)	5.1 Tiempo no reportado (horas)	30 Porcentaje de tiempo no reportado

Tabla C30 Análisis de datos del Metrobús 910

Vehicle:MetrobúsCDMX:910			
2022-10-20 06:37:06 Fecha y hora de inicio de monitoreo	2022-10-20 23:34:28 Fecha y hora final del monitoreo	61042 Periodo de funcionamiento del Metrobús (segundos)	18477 Total de tiempo no reportado (segundos)
88.47 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.		 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro. (ampliación del 30%)	
16.96 Periodo de funcionamiento del Metrobús (horas)	1.47 Tiempo promedio de actualización (minutos)	5.13 Tiempo no reportado (horas)	30 Porcentaje de tiempo no reportado

C.2 Resultados obtenidos para el monitoreo de los datos de la congestión vehicular de la CDMX

En las tablas Tabla 31 a la Tabla 60 se presentan los resultados obtenidos al monitorear los datos de la congestión vehicular de la CDMX. Se utilizan 30 segmentos elegidas al azar.

Tabla C31 Análisis de datos del segmento 4494-

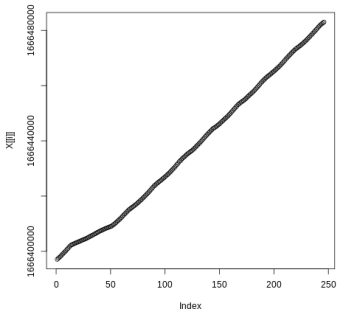
TrafficFlowObserved:CDMX:4494-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8930 Total de tiempo no monitoreado (segundos)
349.26 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.82 Tiempo promedio de actualización (minutos)	2.48 Tiempo no monitoreado (horas)	10.39 Porcentaje de tiempo no monitoreado

Tabla C32 Análisis de datos del segmento 11392-

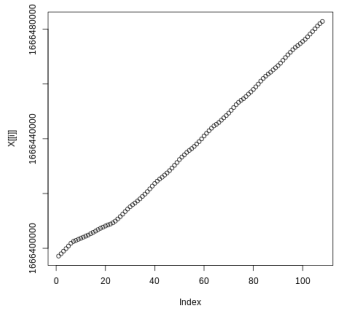
TrafficFlowObserved:CDMX:11392-			
2022-10-22 00:06:02 Fecha y hora de inicio de monitoreo	2022-10-22 23:54:02 Fecha y hora final del monitoreo	85680 Total de tiempo monitoreado (segundos)	8747 Total de tiempo no monitoreado (segundos)
793.33 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.8 Tiempo de monitoreo (horas)	13.22 Tiempo promedio de actualización (minutos)	2.43 Tiempo no monitoreado (horas)	10.21 Porcentaje de tiempo no monitoreado

Tabla C33 Análisis de datos del segmento 12952-

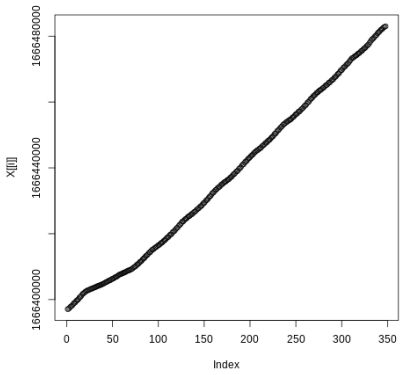
TrafficFlowObserved:CDMX:12952-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	11701 Total de tiempo no monitoreado (segundos)
246.89 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	3.25 Tiempo no monitoreado (horas)	10.39 Porcentaje de tiempo no monitoreado

Tabla C34 Análisis de datos del segmento 4505-

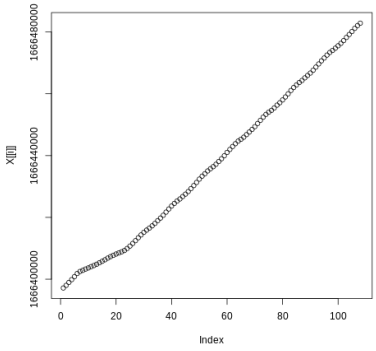
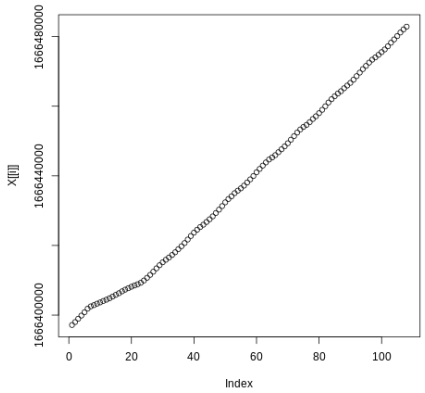
TrafficFlowObserved:CDMX:4505-			
2022-10-22 00:06:02 Fecha y hora de inicio de monitoreo	2022-10-22 23:54:02 Fecha y hora final del monitoreo	85680 Total de tiempo monitoreado (segundos)	8747 Total de tiempo no monitoreado (segundos)
793.33 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.8 Tiempo de monitoreo (horas)	13.22 Tiempo promedio de actualización (minutos)	2.43 Tiempo no monitoreado (horas)	10.21 Porcentaje de tiempo no monitoreado

Tabla C35 Análisis de datos del segmento 14519+

TrafficFlowObserved:CDMX:14519+			
2022-10-22 00:06:02 Fecha y hora de inicio de monitoreo	2022-10-22 23:54:02 Fecha y hora final del monitoreo	85680 Total de tiempo monitoreado (segundos)	8747 Total de tiempo no monitoreado (segundos)
793.33 Tiempo promedio de actualización (segundos)			

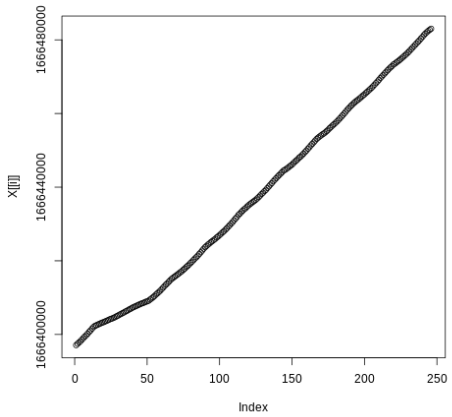


Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.

23.8 Tiempo de monitoreo (horas)	13.22 Tiempo promedio de actualización (minutos)	2.43 Tiempo no monitoreado (horas)	10.21 Porcentaje de tiempo no monitoreado

Tabla C36 Análisis de datos del segmento 13125-

TrafficFlowObserved:CDMX:13125-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
243.4 Tiempo promedio de actualización (segundos)			



Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.

23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C37 Análisis de datos del segmento 4500-

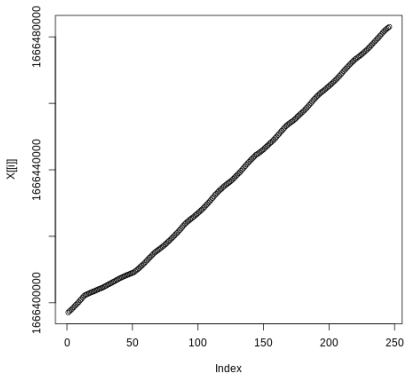
TrafficFlowObserved:CDMX:4500-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
258.79 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C38 Análisis de datos del segmento 4523-

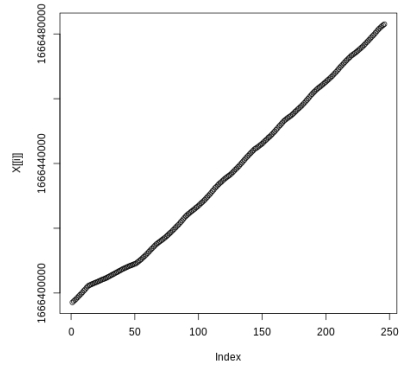
TrafficFlowObserved:CDMX:4523-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
258.79 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C39 Análisis de datos del segmento 13213-

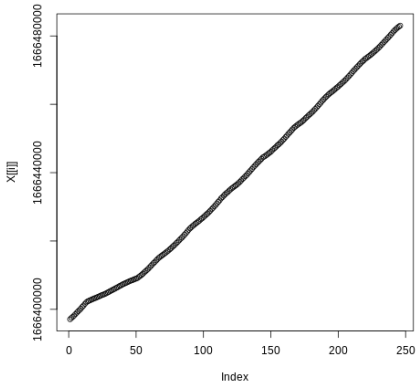
TrafficFlowObserved:CDMX:13213-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
243.4 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C40 Análisis de datos del segmento 4495-

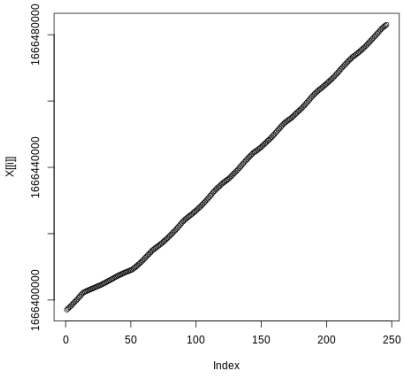
TrafficFlowObserved:CDMX:4495-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
349.26 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.82 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C41 Análisis de datos del segmento 4296-

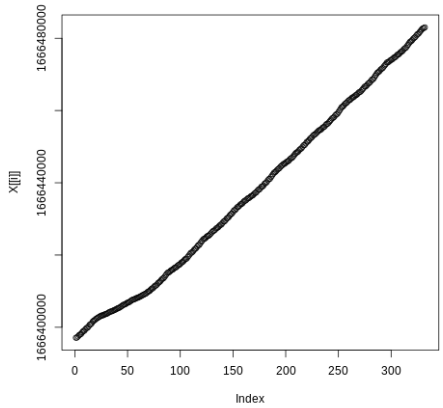
TrafficFlowObserved:CDMX:4296-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
258.79 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C42 Análisis de datos del segmento 13110-

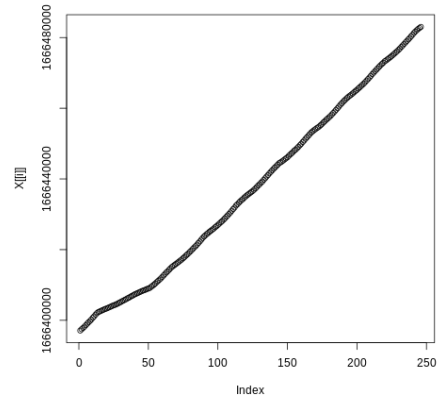
TrafficFlowObserved:CDMX:13110-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
243.4 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C43 Análisis de datos del segmento 4107-

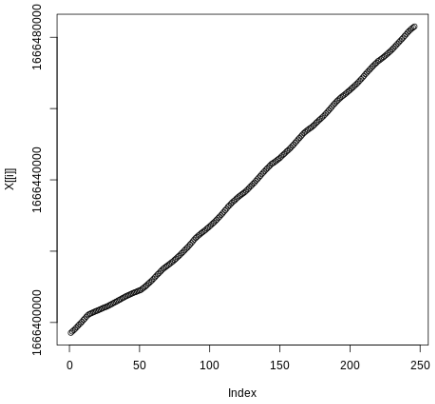
TrafficFlowObserved:CDMX:4107-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8930 Total de tiempo no monitoreado (segundos)
349.26 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.82 Tiempo promedio de actualización (minutos)	2.48 Tiempo no monitoreado (horas)	10.39 Porcentaje de tiempo no monitoreado

Tabla C44 Análisis de datos del segmento 13437-

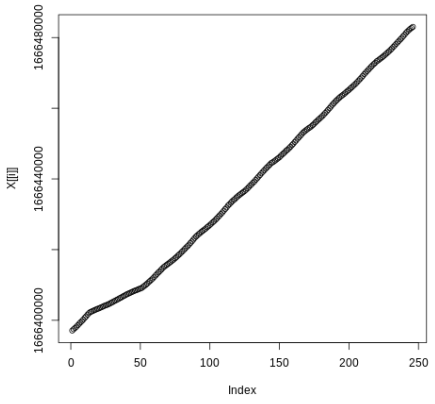
TrafficFlowObserved:CDMX:13437-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 00:05:03 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8930 Total de tiempo no monitoreado (segundos)
349.26 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.82 Tiempo promedio de actualización (minutos)	2.48 Tiempo no monitoreado (horas)	10.39 Porcentaje de tiempo no monitoreado

Tabla C45 Análisis de datos del segmento 4363-

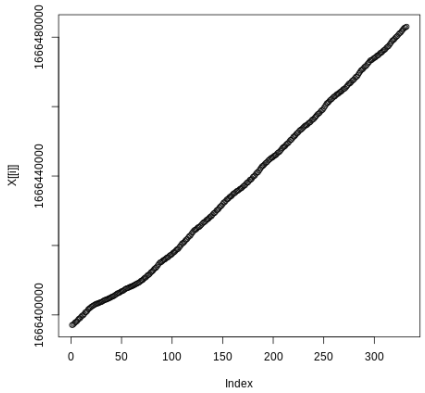
TrafficFlowObserved:CDMX:4363-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
258.79 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C46 Análisis de datos del segmento 4284+

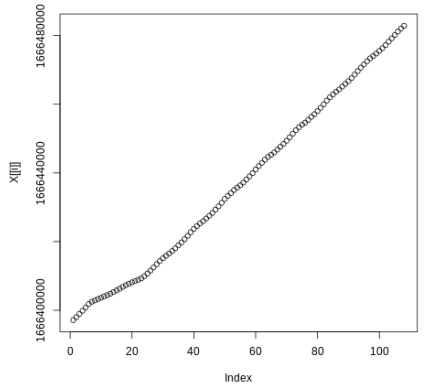
TrafficFlowObserved:CDMX:4284+			
2022-10-22 00:06:02 Fecha y hora de inicio de monitoreo	2022-10-22 23:54:02 Fecha y hora final del monitoreo	85680 Total de tiempo monitoreado (segundos)	8747 Total de tiempo no monitoreado (segundos)
793.33 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.8 Tiempo de monitoreo (horas)	13.22 Tiempo promedio de actualización (minutos)	2.43 Tiempo no monitoreado (horas)	10.21 Porcentaje de tiempo no monitoreado

Tabla C47 Análisis de datos del segmento 13034-

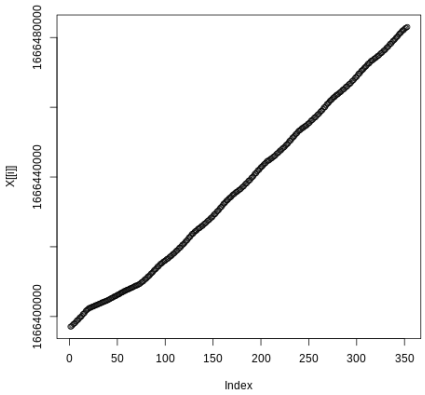
TrafficFlowObserved:CDMX:13034-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
243.4 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C48 Análisis de datos del segmento 13169-

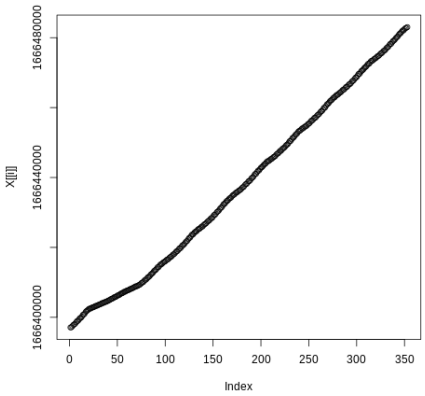
TrafficFlowObserved:CDMX:13169-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
243.4 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C49 Análisis de datos del segmento 4474+

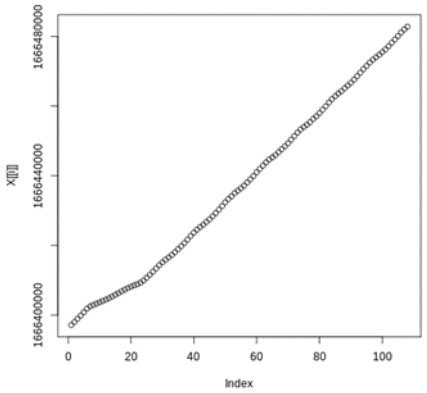
TrafficFlowObserved:CDMX:4474+			
2022-10-22 00:06:02 Fecha y hora de inicio de monitoreo	2022-10-22 23:54:02 Fecha y hora final del monitoreo	85680 Total de tiempo monitoreado (segundos)	8747 Total de tiempo no monitoreado (segundos)
793.33 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.8 Tiempo de monitoreo (horas)	13.22 Tiempo promedio de actualización (minutos)	2.43 Tiempo no monitoreado (horas)	10.21 Porcentaje de tiempo no monitoreado

Tabla C50 Análisis de datos del segmento 5147-

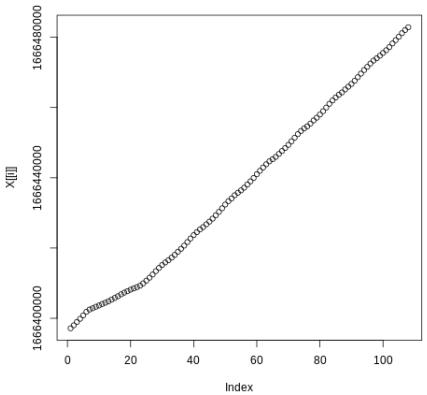
TrafficFlowObserved:CDMX:5147-			
2022-10-22 00:06:02 Fecha y hora de inicio de monitoreo	2022-10-22 23:54:02 Fecha y hora final del monitoreo	85680 Total de tiempo monitoreado (segundos)	8747 Total de tiempo no monitoreado (segundos)
793.33 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.8 Tiempo de monitoreo (horas)	13.22 Tiempo promedio de actualización (minutos)	2.43 Tiempo no monitoreado (horas)	10.21 Porcentaje de tiempo no monitoreado

Tabla C51 Análisis de datos del segmento 4366+

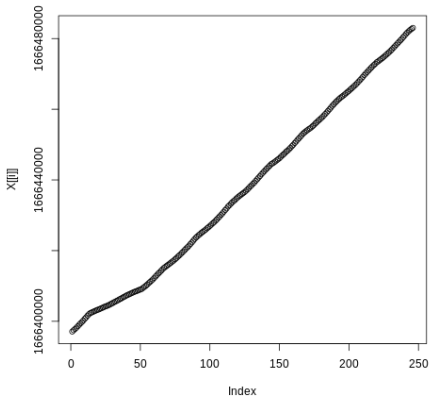
TrafficFlowObserved:CDMX:4366+			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
258.79 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C52 Análisis de datos del segmento 13176-

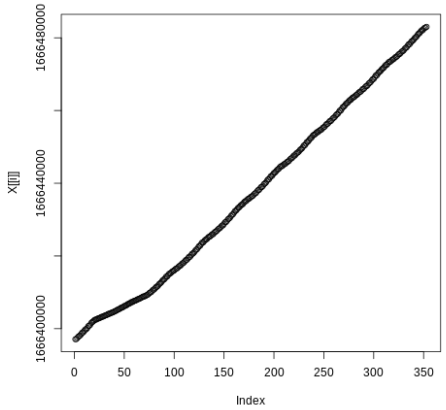
TrafficFlowObserved:CDMX:13176-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
243.4 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C53 Análisis de datos del segmento 4485+

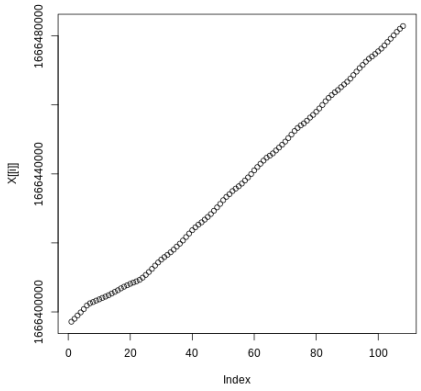
TrafficFlowObserved:CDMX:4485+			
2022-10-22 00:06:02 Fecha y hora de inicio de monitoreo	2022-10-22 23:54:02 Fecha y hora final del monitoreo	85680 Total de tiempo monitoreado (segundos)	8747 Total de tiempo no monitoreado (segundos)
793.33 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.8 Tiempo de monitoreo (horas)	13.22 Tiempo promedio de actualización (minutos)	2.43 Tiempo no monitoreado (horas)	10.21 Porcentaje de tiempo no monitoreado

Tabla C54 Análisis de datos del segmento 4895-

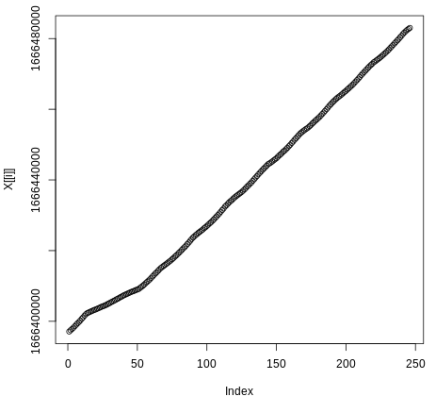
TrafficFlowObserved:CDMX:4895-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
258.79 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C55 Análisis de datos del segmento 4510-

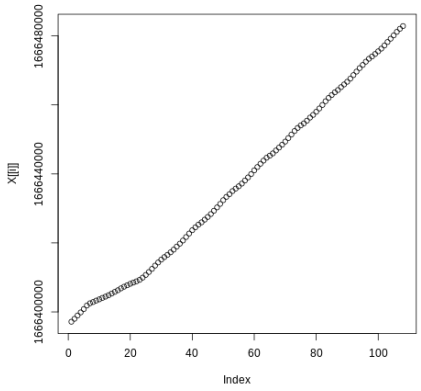
TrafficFlowObserved:CDMX:4510-			
2022-10-22 00:06:02 Fecha y hora de inicio de monitoreo	2022-10-22 23:54:02 Fecha y hora final del monitoreo	85680 Total de tiempo monitoreado (segundos)	8747 Total de tiempo no monitoreado (segundos)
793.33 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.8 Tiempo de monitoreo (horas)	13.22 Tiempo promedio de actualización (minutos)	2.43 Tiempo no monitoreado (horas)	10.21 Porcentaje de tiempo no monitoreado

Tabla C56 Análisis de datos del segmento 12969+

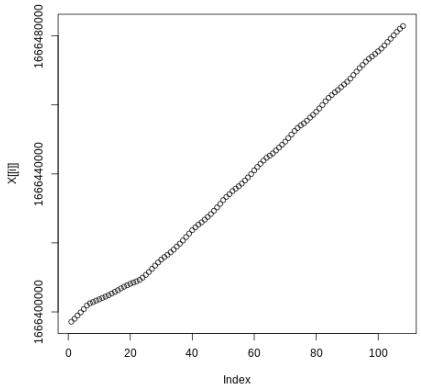
TrafficFlowObserved:CDMX:12969+			
2022-10-22 00:06:02 Fecha y hora de inicio de monitoreo	2022-10-22 23:54:02 Fecha y hora final del monitoreo	85680 Total de tiempo monitoreado (segundos)	8747 Total de tiempo no monitoreado (segundos)
793.33 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.8 Tiempo de monitoreo (horas)	13.22 Tiempo promedio de actualización (minutos)	2.43 Tiempo no monitoreado (horas)	10.21 Porcentaje de tiempo no monitoreado

Tabla C57 Análisis de datos del segmento 4237+

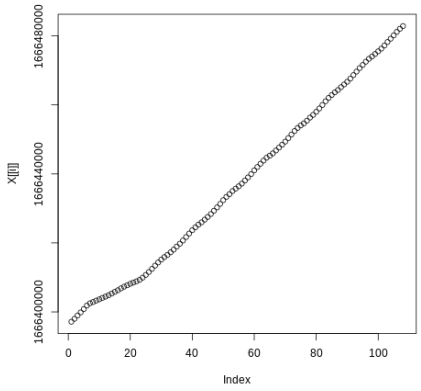
TrafficFlowObserved:CDMX:4237+			
2022-10-22 00:06:02 Fecha y hora de inicio de monitoreo	2022-10-22 23:54:02 Fecha y hora final del monitoreo	85680 Total de tiempo monitoreado (segundos)	8747 Total de tiempo no monitoreado (segundos)
793.33 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.8 Tiempo de monitoreo (horas)	13.22 Tiempo promedio de actualización (minutos)	2.43 Tiempo no monitoreado (horas)	10.21 Porcentaje de tiempo no monitoreado

Tabla C58 Análisis de datos del segmento 13165-

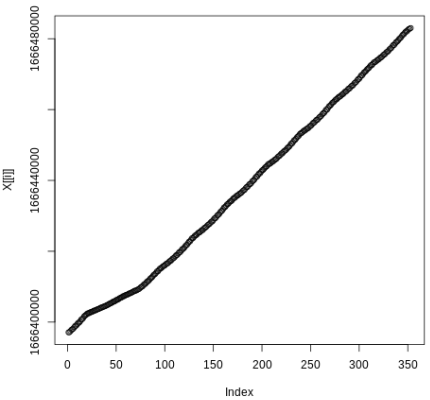
TrafficFlowObserved:CDMX:13165-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
243.4 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C59 Análisis de datos del segmento 12947-

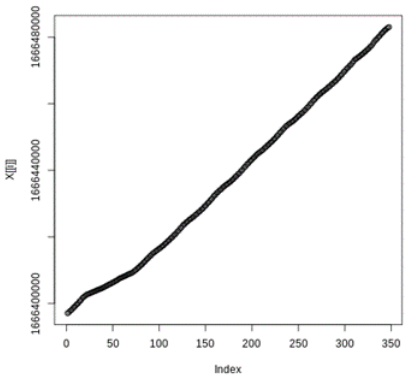
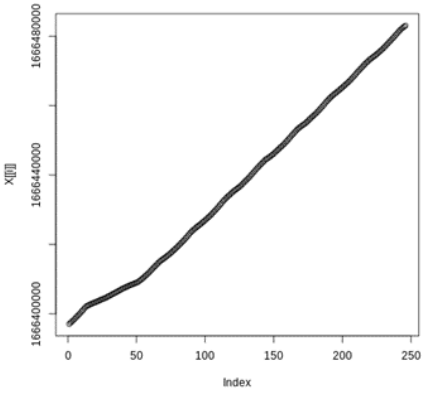
TrafficFlowObserved:CDMX:12947-			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8248 Total de tiempo no monitoreado (segundos)
246.89 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.5 Tiempo promedio de actualización (minutos)	2.29 Tiempo no monitoreado (horas)	9.6 Porcentaje de tiempo no monitoreado

Tabla C60 Análisis de datos del segmento 4219+

TrafficFlowObserved:CDMX:4219+			
2022-10-22 00:05:03 Fecha y hora de inicio de monitoreo	2022-10-22 23:57:02 Fecha y hora final del monitoreo	85919 Total de tiempo monitoreado (segundos)	8930 Total de tiempo no monitoreado (segundos)
349.26 Tiempo promedio de actualización (segundos)			
 Comportamiento de actualización de datos según el tiempo. En el círculo rojo se observan los puntos donde no existió registro.			
23.87 Tiempo de monitoreo (horas)	5.82 Tiempo promedio de actualización (minutos)	2.48 Tiempo no monitoreado (horas)	10.39 Porcentaje de tiempo no monitoreado