Otero Dacasa Marcos

ESTUDIO TÉCNICO DE LA FACTIBILIDAD Y PROPUESTA TECNOLÓGICA PARA SISTEMAS DE MONITOREO EN LOS SISTEMAS DE TRANSPORTE PÚBLICO

Ingeniería en Telecomunicaciones

Universidad Nacional Autónoma de México
México 2013


UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ingeniería

TESIS

ESTUDIO TÉCNICO DE LA FACTIBILIDAD Y PROPUESTA TECNOLÓGICA PARA SISTEMAS DE MONITOREO EN LOS SISTEMAS DE TRANSPORTE PÚBLICO

Que para obtener el título de Ingeniero en Telecomunicaciones

Presenta

Otero Dacasa Marcos

Dr. Carlos Gershenson García

Ciudad Universitaria, México- 4 de febrero de 2014

Índice general

Índice de figuras

Índice de tablas

Glosario

1. **Atomicity, Consistency, Isolation y Durability (ACID):** En teoría de computación se refiere a una serie de propiedades que deben de cumplir todos los sistemas y procesos que manejen información. Esta forma de construcción es especialmente importante en bases de datos donde la integración de la información debería estar garantizada en todo momento.

2. **Administrador de bases de datos (DBMS):** Es un sistema en software que permite la definición, creación, petición, actualización y administración de bases de datos.

3. **Agencia: Institución de la administración pública que se encarga de ejecutar y administrar algún servicio público**

4. **Agile Develoment:** Es una serie de métodos basados en desarrollos iterativos e incrementales, en donde los los requerimientos y las soluciones evolucionan a través de colaboración de diferentes equipos.

5. **Applets:** Aplicaciones especializadas una operación que funcionan dentro de programas más grandes o como programas independientes.

6. **Application Programming Interface (API):** Serie de rutinas, protocolos y herramientas que establecen cómo interactúa un sistema con sistemas externos.

7. **Arreglos:** Estructuras de datos ordenadas que permiten acceder elementos dentro de ella a través de índices numéricos.

8. **Arqueo-tipo:** Este término se refiere a una forma pura que funciona como la característica fundamental de otra cosa. En el caso de este trabajo, representa la categoría de los recursos dentro de un servidor dependiendo de su estructura.

9. **Atomicity:** Esta propiedad requiere que los procesos sean "todo o nada". Esto significa que si la transacción falla el estado de la información debe poderse revertir a su forma original.

10. **Automatic Vehicle Location (AVL):** Todo aquel sistema que permite conocer su posición en tiempo real, generalmente usando un GPS y un sistema de transmisión a través de un módem inalámbrico.

11. **Bases de datos relacionales:** Es un sistema de manejo de bases de datos que está basado en modelos relacionales.

12. **Bases de datos de objetos relacionales (ORDBMS):** Este tipo de base de datos puede guardar objetos más complejos en sus tablas relacionales que sólo fechas, números y texto. Permite al usuario definir tipos de datos, nuevas funciones y operadores para manipular estos datos.

13. **Bases de datos NoSQL:** Son bases de datos que permiten guardar información y leerla utilizando modelos menos consistentes que los que usa una base de datos relacional.

14. **Big Data:** Es un término utilizado para describir un conjunto de datos tan grande y complejo que es difícil procesarlo con técnicas convencionales.

15. **Binary Large Object (BLOB):** Es un conjunto de datos en forma binaria que se encuentran guardados dentro de una base de datos. Consisten principalmente de imágenes, videos, audio y otras formas de multimedia.

16. **Bussines Activity Monitor (BAM):** Permite revisar todos lose servicios en un sistema distribuido heterogéneo a través de una interfaz XML.

17. **C10K:** Se refiere al problema de optimizar sockets en una red para poder manejar una gran cantidad de conexiones al mismo tiempo (Superiores a las decenas de miles).

18. **Carriage return and Line Feed (CRLF):** Es una secuencia de caracteres representando una nueva linea con recorrido de carro. Dependiendo del sistema de codificación esta secuencia puede cambiar.

19. **Concurrencia:** En computación, la concurrencia es la propiedad de los sistemas que permiten que múltiples procesos sean ejecutados al mismo tiempo, y que potencialmente puedan interactuar entre sí.

20. **Consistency:** Esta propiedad se asegura que cada una de las transacciones o procesos que se realicen sobre la información darán como resultado un sólo estado bien conocido que se conoce como “estado válido“.

21. **Content Delivery Network (CDN):** Red de distribución de contenido de gran escala constituido de múltiples servidores en centros de datos a través de internet. Su objetivo es entregar contenido a los usuarios finales en el menor tiempo posible.

22. **Colección:** Dentro de una API una colección se puede entender como un conjunto de documentos o un directorio de documentos. Esta palabra no debe ser confundida con las colecciones que se encuentran dentro de los lenguajes de programación ya que propiamente hablando, éstos últimos son una estructura de datos que se encuentran contiguos dentro de la memoria.

23. **Controlador:** Es un documento que modela una acción que pueden verse como métodos en un lenguaje de programación, a los cuales se le puede enviar argumentos y retorna variables.

24. **Cross Site Scripting (XSS):** Es un ataque en la que un atacante inyecta un script del lado del cliente permitiéndole traspasar controles de acceso.

25. **Cython:** Es un lenguaje que permite construir extensiones en el lenguaje C para python de una manera sencilla.

26. **Disparador en bases de datos (Triggers:)** Código que es automáticamente ejecutado en respuesta a un evento dentro de la base de datos. Este código debe de asegurarse de manejar ACID.

27. **Dispositivo:** Los dispositivos, dentro de éste trabajo de tesis, serán definidos como componentes electrónicos que leen o escriben datos en medios o soportes de almacenamiento a través de una red de datos.

28. **Documento (URL):** Es un concepto singular que puede estar relacionado con un objeto o un registro en una base de datos. Este tipo de recurso puede contener ligas hacia otros recursos, así como sus valores propios. Esta estructura mínima es la base de otro tipo de arqueo-tipos.

29. **Document Object Model (DOM):** Es una convención independiente del lenguaje y la plataforma para representar e interactuar con objetos en documentos basados en etiquetas como XML y HTML.

30. **Durability:** Implica que una vez que la transacción o la operación sobre la información ha finalizado, esta debe de perdurar no importando qué sucedan eventualidades como cortes eléctricos, problemas en el servidor o en el sistema operativo.

31. **Extensible Markup Language (XML):** Es un lenguaje basado en etiquetes desarrollados por la W3C cuya misión es reducir la complejidad del estándar SGML y crear una representación de datos legible para los humanos.

32. **Enteprise Service Bus (ESB):** Es la parte de SOA que permite tanto a servicios internos como a externos comunicarse entre ellos. Este bus permite independencia entre los servicios a costa de pérdida en la velocidad de respuesta del sistema.

33. **European Petroleum Survey Group (EPSG):** Es una organización conformada por empresas que comercian con petroleo y gas en la Unión Europea y el Norte de África.

34. **European Petroleum Survey Group ID (EPSG ID):** Las empresas que conforman este consorcio necesitaban resolver el principal problema cuando se buscan reservas de

petroleo: posición a escala global. Las empresas utilizaban cada una sus escalas y sistemas de coordenadas propios.

La solución fue crear una tabla con las diferencias entre cada escala y la información necesaria para poder convertir de una escala a otra sin una pérdida significante de información.

35. **Feed:** Mecanismo utilizado por los usuarios para recibir información actualizada proveniente de diferentes fuentes. Son comúnmente usados en aplicaciones en tiempo real a través de la web.

36. **Framework:** La palabra inglesa "framework"(marco de trabajo) define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

37. **Geographic Information Systems (GIS):** Todo aquel sistema que permite crear, ordenar, organizar, analizar y manejar información espacial y sus atributos.

38. **Global Position Systems (GPS):** Sistema de navegación por satélite que permite saber la posición y la hora en todas las condiciones climáticas en cualquier lugar de la tierra en donde no exista una obstrucción de la señal de al menos 3 o más satélites.

39. **Google Transit Feed Specification (GTFS):** Es un estándar propuesto por la empresa estadounidense Google, el cual da un formato que se puede utilizar para guardar y compartir información geográfica.

40. **GTFS in real Time (GTFST):** Extensión del estándar GTFS que propone un formato para publicar información geográfica en tiempo real.

41. **Hypertext Markup Languaje (HTML):** Es un estándar derivado del estándar SGML cuyo propósito principal es la definición de una estructura básica para la creación de páginas web.

42. **Idempotencia:** Es la habilidad de los servicios para manejar mensajes iguales que llegan más de una vez, de manera que no se realice la misma operación múltiples veces.

43. **Internet Assigned Numbers Authority (IANA):** Organismo Estadounidense encargado de la asignación de la claves IP, las zonas de rutas, el manejo de nombres de dominio y URL, así como otros símbolos y números relacionados con los protocolos de internet.

44. **Isomorfismo:** Propiedad de las representaciones de datos que establecen que dada una representación de información en uno de estos formatos, es posible crear otro formato que contenga lo mismo en el mismo orden.

45. **IronPython:** Implementación del lenguaje Python que es compatible con el framework .NET y Mono.

46. **Isolation:** Esta propiedad asegura que el estado final de la información es el mismo ya sea que las instrucciones sean ejecutadas en serie o en paralelo.

47. **Java ByteCode:** Instrucciones del lenguaje de programación Java transformados en opcodes a través de un compilador.

48. **Java Server Faces (JSF):** Es una implementación en el lenguaje Java que permite crear interfaces de usuarios especializadas en aplicaciones web.

49. **Java Server Pages (JSP):** Tecnología que ayuda a los desarrolladores del lenguaje Java a crear aplicaciones web que utilicen contenido dinámico.

50. **Java Virtual Machine (JVM):** Máquina Virtual con la capacidad de ejecutar instrucciones del lenguaje Java bytecode.

51. **Jython:** Jython es una integración de Python con el lenguaje Java que le permite llamar cualquier clase de Java. Esta variante se diferencia del lenguaje original debido a que es compilado y corre sobre la JVM

52. **Máquina Virtual:** Software que emula la implementación de una computadora en código con la capacidad de ejecutar códigos al igual que una máquina física.

53. **MyISAM:** Es la herramienta de guardado por defecto utilizado en la base de datos MySQL. Consiste en un sistema de guardado en disco con tres archivos. Los archivos tienen nombres que comienzan con el nombre de la tabla y un extensión que establece que tipo de archivo.

54. **Modelos Relacionales:** En un modelo relación toda la información es presentada en tuplas que forman tablas agrupadas por relaciones con otras tablas.

55. **Multipurpose Internet Mail Extensions:** fueron originalmente diseñados para el intercambio de correos entre empresas de correo electrónico, pero después fueron adoptadas por el estándar HTTP.

56. **Objects-literals:** Representaciones de objetos en los que los atributos y sus valores son escritos y separados por algún símbolo en común.

57. **Opcodes:** Son instrucciones que tienen un byte de largo, pudiendo representar hasta 256 operaciones distintas, aunque algunas pueden requerir parámetros transformándolas en instrucciones multibyte.

58. **Open Geospatial Consortium (OGC):** Es el organismo encargado de crear y regular los estándares para información geográfica y las herramientas que interactúan con ella. Entre sus estándares se encuentran los formatos en que la información debe ser guardada dentro de bases de datos o la estructura y nombre de las sentencias para solicitarla.

59. **Open Source Geospatial Foundation (OSGeo):** Es el organismo que maneja las iniciativas para la creación de herramientas de código fuente abierto que funcionen con información geográfica.

60. **Pypy:** Es un intérprete de python que realiza optimización en tiempo real. Este intérprete intenta ser compatible con el original Cpython.

61. **Proj4:** Es una librería diseñada para realizar conversiones entre proyecciones cartográficas.

62. **Projcs:** Es una librería diseñada para realizar conversiones entre proyecciones cartográficas.

63. **Representational State Transfer (REST):** Es una arquitectura de diseño para sistemas distribuidos de información. En ésta se ignoran los detalles de la implementación de los componentes y la sintaxis del protocolo para enfocarse en los roles de los componentes y su interpretación de los recursos.

64. **Secretaría de Comunicaciones y Transportes (SCT):** Organismo gubernamental encargado de la gestión de obras públicas en temas de transportes y telecomunicaciones.

65. **Secretaria de Transporte y Vialidad (SETRAVI):** Organismo gubernamental que tiene como tara la administración, diseño y construcción de los sistemas de vialidad y transporte dentro del país.

66. **Secreto Oauth2:** Palabra utilizada como semilla en los algoritmos de cifrado dentro del protocolo Oauth2.

67. **Service Oriented Architecture (SOA):** SOA se puede ver como una forma de diseñar sistemas que permite el desarrollo de arquitecturas. No existe actualmente una manera de definirlo, aunque muchos libros lo plantean como una forma de pensar que permite tomar decisiones en arquitectura de software.

68. **Servicio:** Un servicio es un módulo o un conjunto de módulos que presentan una funcionalidad completa o son una aplicación finalizada.

69. **Simplified Wrapper and Interface generator (SWIG):** Es una herramienta de código fuente abierto que permite la conexión de diferentes lenguajes de programación con librerías del lenguaje C++.

70. **Simple Object Access Protocol (SOAP):**

71. **Sistemas de almacenado:** Las URL que apuntan a sistemas que son manejados por el cliente son conocidas como sistemas de almacenado. Este tipo de recursos permiten al cliente trabajar en recursos propios.

72. **Structured Query Language (SQL):** Es un lenguaje de propósito especial diseñado para manejar información dentro de bases de datos relacionales.

73. **Tecnologías de la Información y la Comunicación (TIC):** Es una denominación utilizada para describir a todas las tecnologías que electrónicas especializadas en el procesamiento de información y en la transmisión de esta.

74. **Test driven development (TDD):** Es una técnica de programación en la que el código de testeo se programa primero y después se crear el código del sistema que permite pasar el test.

75. **Transporte Colectivo:** Contempla los servicios prestados en vagonetas, combi, microbuses y en menor medida autobuses.

76. **Vehículo:** Un vehículo es un medio de locomoción que permite el traslado de un lugar a otro. Cuando traslada a animales u objetos es llamado vehículo de transporte, como por ejemplo el tren, el automóvil, el camión, el carro, el barco, el avión, la bicicleta y la motocicleta, entre otros.

Resumen

Los sistemas de información geográfica (GIS[1]) son cada vez más importantes en nuestra sociedad. La posibilidad de mostrar información geográfica en una interfaz hace mucho más sencillo trabajar con ésta y encontrar patrones en ella, con respecto a si esa información se encuentra sólo en papel y mapas comunes.

La demostración de su importancia la podemos ver actualmente en los servicios públicos más comunes como hospitales, secretarías, sistemas de transporte público, entre otros, que requieren que su información sea provista a sus usuarios de manera sencilla, rápida y fácil de entender. Actualmente ya existen varios países que están aprovechando las posibilidades que este tipo de sistemas les brindan.

Sin embargo, la capa de presentación de estos sistemas, que en la mayoría de los casos es un mapa, es apenas uno de los tantos subsistemas interconectados a través de diferentes interfaces (API)[2] que interactúan a través del envío y recepción de una gran cantidad de datos.

Este proyecto presenta un estudio tecnológico para una propuesta que pueda ser utilizada junto con los sistemas de localización automática de vehículos (AVL[3]) que actualmente se utilizan en el sistema transporte público de la zona metropolitana, así como para aquellos sistemas que aún no cuentan con esta tecnología, lo que permitirá unificar los sistemas de control utilizados en diferentes administraciones como el Metrobús, RTP, Metro, etc. Esto se intenta lograr diseñando un sistema de recepción, procesamiento y organización de datos geográficos, basado en sistemas abiertos y estándares internacionales, el cual recibe la posición geográfica del vehículo y la guarda en una base de datos de acceso abierto.

Un sistema centralizado de esta naturaleza reducirá la complejidad de la gestión de estos servicios y permitirá una sincronización entre ellos de una manera más sencilla. De esta manera, la información recopilada permitirá en un futuro procesar la información guardada en las bases de datos y con ello encontrar problemas dentro del transporte o reducir costos mejorando la administración de los sistemas de transporte.

La información será recopilada utilizando el estándar GTFS[4], el cual ya está siendo utilizado actualmente por el gobierno de la Ciudad de México para publicar algunos datos de sus

[1] GIS: Geographic Information Systems
[2] API: Application Programming Interface
[3] AVL: Automatic Vehicle Location
[4] GTFS: General Transit Feed Specification

sistemas[5], que permite subir y consultar información en un formato común a través de diferentes servidores alrededor del mundo, así como trabajar en colaboración con otras instituciones que lo utilicen. Al mismo tiempo, es posible a través de este estándar, acceder a información abierta por otros gobiernos y empresas alrededor del mundo para complementar la propia y así poder realizar estudios sobre los sistemas de transporte.

El prototipo construido en este trabajo, consistente de un servidor y una base de datos geográfica, permite guardar y publicar información en GTFS, importar de fuentes externas a la base de datos y exportar de la base de datos hacia fuentes externas a través del estándar GTFSRT[6]. Éste prototipo es diseñado con el objetivo de demostrar cómo debe de funcionar un sistema de estas características y por ello no debe de ser utilizado como una solución final o en fase de producción.

Finalmente, se describen los actuales problemas de tráfico y de administración que se encuentran dentro de la zona metropolitana actualmente, se analizan los efectos que estos tienen con otros sistemas de manera indirecta y se cierra éste trabajo de tesis con las conclusiones y sugerencias futuras para el proyecto basado en la experiencia que se obtuvo a lo largo del desarrollo del proyecto.

[5] http://datosabiertos.df.gob.mx/index.php/busqueda-por-dependencia/81-transporte- y-vialidad/787-base-de-datos-abiertos-de-transporte-del-distrito-federal

[6] GTFSRT: GTFS en tiempo real

Objetivos de la Tesis

El presente trabajo de tesis tiene un objetivo general y cuatro objetivos específicos. El objetivo general es proponer y analizar las tecnologías necesarias para la construcción de un sistema de monitoreo para el sistema de transporte público de la zona metropolitana del valle de México (ZMVM); así como diseñar un prototipo que proponga una arquitectura de conexión entre estas tecnologías. Este prototipo se diseña intentado demostrar la posibilidad y viabilidad tecnológica del sistema, además de proponerlo como base para el desarrollo de un proyecto más grande a futuro. Así mismo, intentará demostrar cómo funcionan e interactúan las tecnologías que se propongan a lo largo del proyecto, todo esto sin tratar de diseñar un sistema que contenga más allá de lo necesario para poder lograr los objetivos presentados aquí.

Para alcanzar este objetivo general, se han establecido cuatro objetivos específicos necesarios, en todo caso, para la consecución del objetivo general de este trabajo. El primer objetivo es el estudio de la problemática actual que existe en los sistemas de transporte público de la zona metropolitana. Se intenta, en primer lugar, describir el sector desde una perspectiva conjunta entre la ingeniería y la política, enfatizando especialmente en los problemas que afectan directamente a la eficiencia de los sistemas de transporte y que se deben tomar en cuenta en la realización del proyecto; sin embargo, también se trata de analizar los problemas indirectos que existen y que no tienen un origen en la ingeniería, como son los problemas causados por leyes o mala administración. El estudio de estos problemas deberá permitir identificar y analizar las tecnologías iniciales necesarias.

El segundo objetivo específico es la propuesta de una arquitectura de conexión entre los componentes que sea viable y modular, además de ser fácil de mantener y escalar a largo plazo. Para ello es necesario la identificación de cada una de las tecnologías, los protocolos y estándares que se utilizarán a lo largo del sistema. Estos protocolos, a su vez, van a condicionar la utilización de los módulos y las conexiones entre éstos, así como los flujos de información que existan.

El tercer objetivo específico es el diseño de un prototipo que ayude a comprobar las soluciones propuestas, estudiar la interacción entre las tecnologías, así como servir de base en el desarrollo de proyectos futuros. Para ello, es necesario construir un sistema que permita estudiar la interacción entre estos sistemas. El objetivo de éste prototipo es mostrar los procesos de construcción de la base de datos, los servidores, las funciones y métodos necesarios. Estudiar problemas que pueden llegar a ocurrir dentro del sistema final y que son difíciles de encontrar

si sólo se plantea de una manera teórica, así como permitir el probar los sistemas que puedan interactuar con este y entender los requerimientos básicos para su interoperabilidad. Al mismo tiempo servirá para poder proponer nuevas tecnologías que se podrán utilizar en el futuro y que podrían resolver los problemas que se puedan ir encontrando.

Finalmente, el cuarto objetivo específico es analizar, entender y proponer protocolos y canales de comunicación para realizar una comunicación correcta entre los sistemas, así como permitir escalar el sistema de la manera más sencilla. En este sentido, se trata de entender que desde una perspectiva general, este proyecto puede ser modelado como un sistema de comunicaciones, en el cual se tiene un transmisor (Receptores GPS y sistemas de seguimiento en los vehículos), un canal de comunicación (Internet, Red Telefónica) y un receptor (Servidor y Bases de datos). Esto implica que, al igual que cualquier otro sistema de comunicación, deben de existir estándares y protocolos de comunicación que todos los elementos del sistema entiendan. Sin embargo, también hereda los problemas que estos sistemas tienen, como la posibilidad de que una configuración incorrecta no permita la comunicación, problemas de seguridad informática, entre otros que se intentarán plantear y dar propuestas para su solución en este trabajo.

Introducción

El actual crecimiento de las ciudades alrededor del mundo se está acelerando. La organización mundial de la salud (OMS) establece que por primera vez en la historia la mayor parte de la población mundial vive en ciudades y no se espera que esto cambie en un futuro. En 1990, menos del 40 % de la población en el mundo vivía en ciudades, sin embargo, en 2010 esa proporción ya había pasado el 50 % y se espera que para el 2030 el 60 % de la población ya esté viviendo en ciudades. No se espera que se invierta esta tendencia en un futuro cercano ya que se pronostica que para 2050 esta proporción llegará al 70 % [16].

La zona metropolitana del valle de México (ZMVM) no es una excepción, y en los últimos años a experimentado un aumento de la población que está generando enormes presiones sobre el transporte público y las vías de transporte en la ciudad.

En los últimos 60 años, el crecimiento de la zona metropolitana del valle de México ha eliminado todas las teorías que planteaban que el problema de altas densidades de población sólo se localizaría en países desarrollados. En los años 80 se esperaba que el Valle de México se convirtiera en la zona urbana más poblada del mundo. Un número diferente de factores han impedido que esto ocurra como son la caída de las tasas de nacimiento y los problemas causados por la dificultad de manejar la escalabilidad de la ciudad. En el mapa de la figura 1 se puede ver la evolución de los que es ahora la zona metropolitana la cual ya cubre una zona mayor al Distrito Federal.

Los censos del 2010 realizados por la INEGI [18] demuestran que más del 90 % de la población del área urbana vive en lo que antes se conocían como los suburbios como se muestra en la figura 2

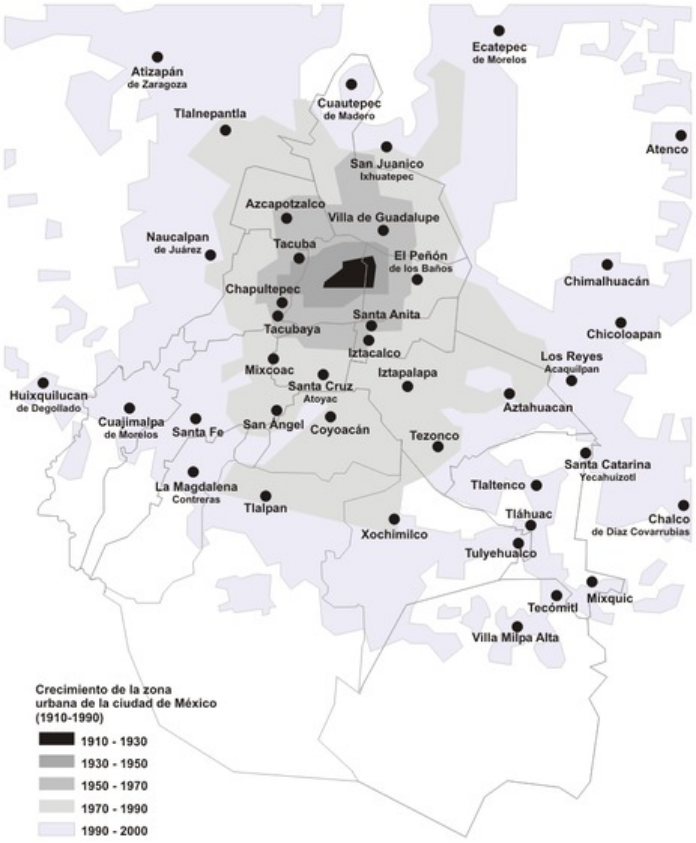

Figura 1: Crecimiento del valle de México entre 1910 al 2000

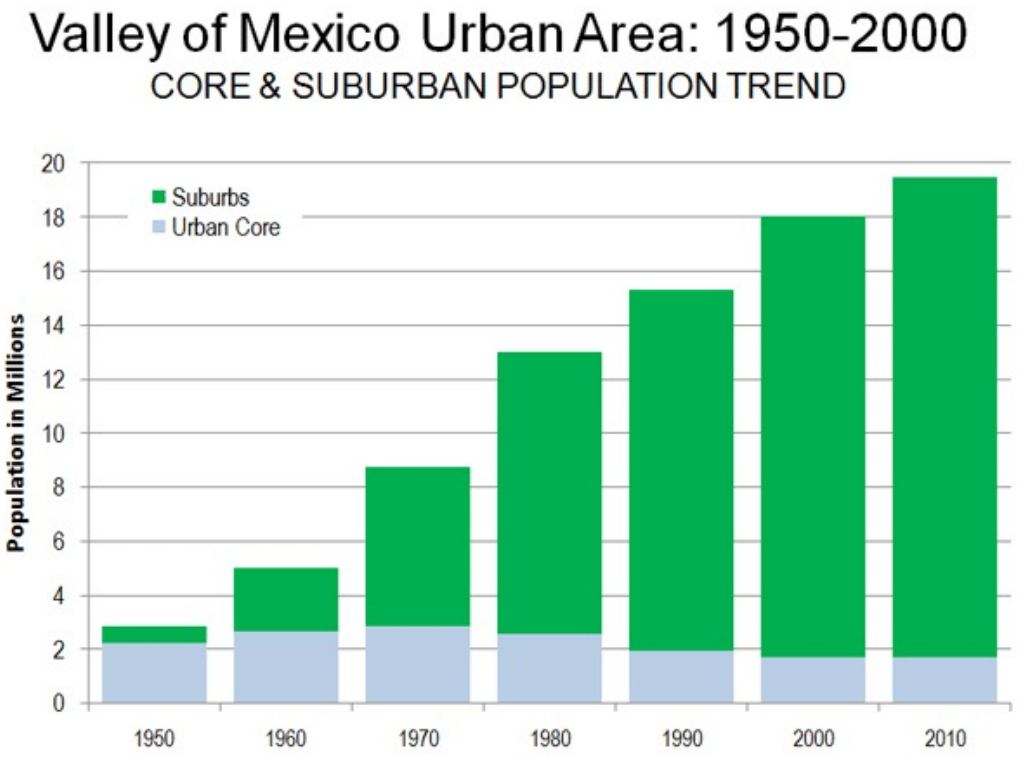

Figura 2: Crecimiento de la población del valle de México [18]

Esta tendencia de la ciudad hacia una dispersión no ha disminuido. Actualmente la Zona metropolitana del Valle de México la conforman las 16 delegaciones del Distrito Federal, 60 municipios del Estado de México y en 2005 se lanzó una proposición para agregar 28 municipios del estado de Hidalgo dentro de esta zona.

Este esparcimiento ha obligado a dividir la zona metropolitana en 5 secciones:

1. **Núcleo urbano:** Abarca la mancha urbana de la zona metropolitana antes de 1994.
2. **El balance urbano:** La zona de la Ciudad de México que se encuentra fuera del Núcleo urbano pero dentro de los límites del Distrito Federal.
3. **Municipios de anillo interno:** Son todos los municipios del Estado de México que comparte frontera con el Distrito Federal.
4. **Municipios del anillo externo:** Son todos los municipios restantes del Estado de México.
5. **Municipios del estado de Hidalgo:** Incluyen los 28 municipios que se propusieron como unión.

En la figura 3 se puede ver el porcentaje de crecimiento que cada una de las cinco secciones ha aportado a la población total de la zona metropolitana entre los años 2000 y 2010.

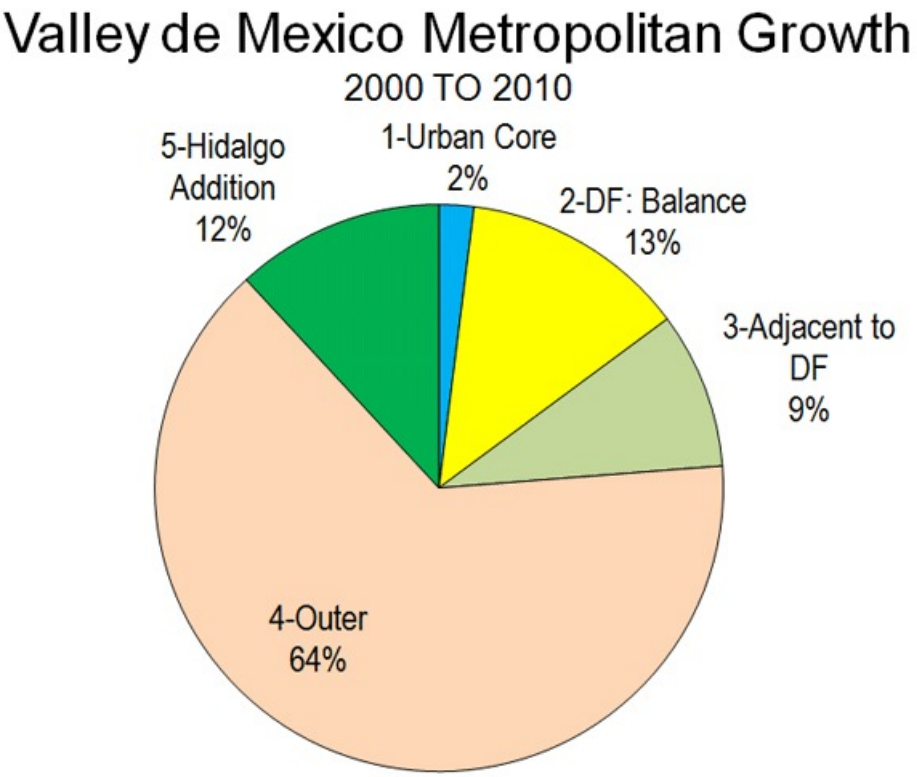

Figura 3: Aportación, en porcentaje de población, de cada una de las secciones [18]

Al mismo tiempo este crecimiento repercute en la población a nivel nacional. En la figura 4 podemos ver la evolución que ha tenido la población del país desde el año 2003.

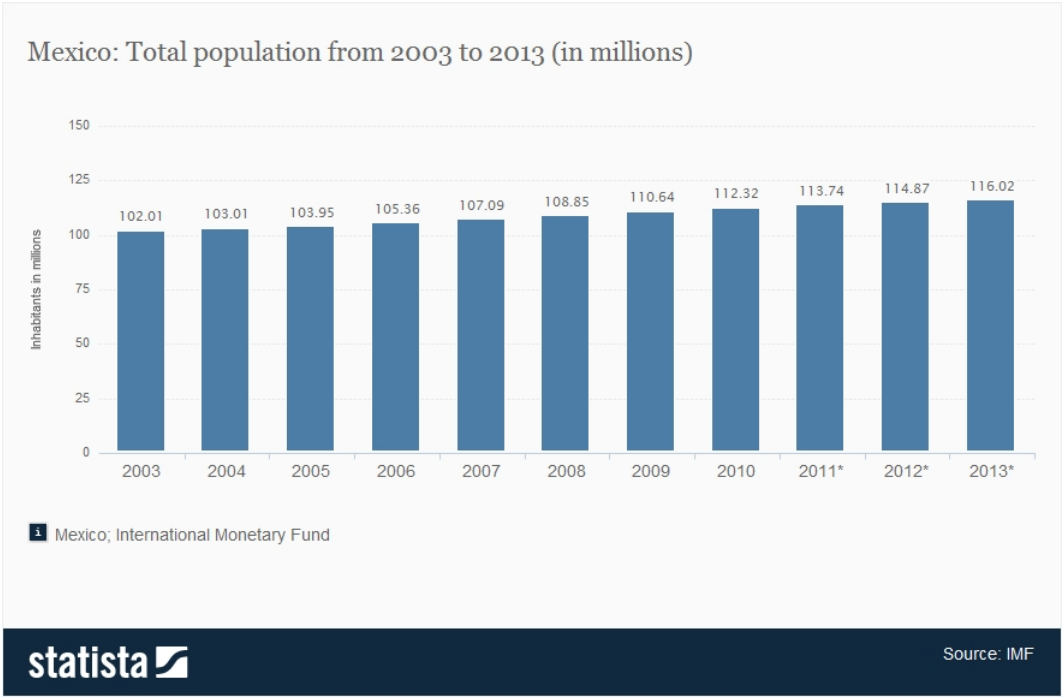

Figura 4: Población en México entre 2003 y 2013 [17]

Este crecimiento lleva consigo una mayor densidad urbana y un mayor número de personas que requieren transportarse entre dos puntos, la mayoría de las personas que viven en las zonas municipales trabajan en el núcleo urbano, generando que los sistemas de transporte apenas pueden soportar la demanda de esta rápida expansión. Desde una perspectiva social esto tiene una gran cantidad de repercusiones como el aumento en el tiempo que la gente pasa viajando de un punto a otro o los problemas de salud que se genera por el estrés que la vida actual está generando en las personas.

La administración de un sistema de este tamaño es cada vez más difícil por la cantidad de información que se debe procesar y la velocidad a la que ésta es generada. Gracias a la rápida evolución de las tecnologías de la información y la comunicación (TIC), es posible procesar una gran cantidad de datos procedentes de miles de sitios casi en tiempo real, lo cual permite gestionar sistemas de gran complejidad como son los sociales, y sus interacciones con subsistemas como los de transporte público. Con esto se logrará disminuir la complejidad de la administración de éstos sistemas y por ello el costo de que estos generan.

A lo largo de este trabajo se diseñará un sistema que aproveche esta habilidad de las TICs para ayudar en la gestión y control del transporte público en la zona metropolitana y que proporcione ésta información a universidades, al gobierno y a la sociedad. Estos podrán utilizar ésta información para crear aplicaciones que permitan mejorar la calidad del transporte público.

Este trabajo ha sido repartido a lo largo de 5 capítulos cada uno dedicado a uno de los procesos que se utilizan dentro de las metodologías Agile Development que definen algunas fases importantes en el desarrollo de un proyecto:

1. **Estudio y análisis del problema:** Estudiado en el capítulo 1
2. **Estudio de las tecnologías a utilizar:** Estudiado en el capítulo 2
3. **Diseño del prototipo y arquitectura:** Estudiado en el capítulo 3
4. **Construcción del sistema:** El capítulo 4 y 5 abordan éste punto
5. **Testeo y retroalimentación:** Esto no se encuentra documentado en un capítulo en particular, aunque el testeo del programa se aborda a lo largo del trabajo a través de capturas de pantalla donde se le muestra funcionando.

En la figura 5 se puede ver el proceso de Agile Development.

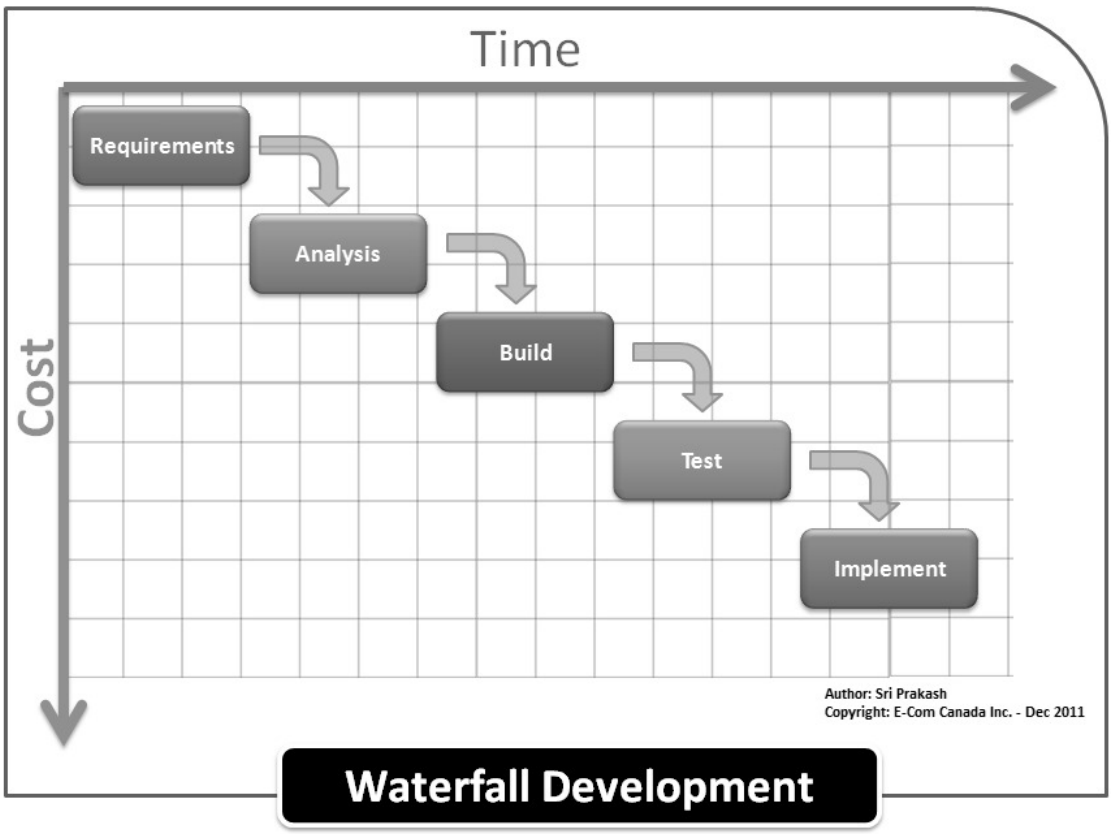

Figura 5: Resumen gráfico del proceso de Agile Development

En el primer capítulo se describe el problema de tránsito actual en la Zona Metropolitana del valle de México, el alcance del proyecto y las limitaciones que puede presentar.

En el capítulo 2 se presentan la metodología a utilizar describiendo cómo ésta es aplicada dentro del proceso del desarrollo de software, las herramientas de desarrollo para el diseño y el estudio del arte de las tecnologías.

Se intenta justificar el lenguaje de programación que se utilizará en el proyecto, se realiza un análisis de diferentes dispositivos de seguimiento que pueden ser utilizados y finalmente se proponen la base de datos y el servidor partiendo de algunos requerimientos básicos que necesita

el proyecto. Todo esto se realiza a través de la propuesta de diferentes opciones para cada una de las tecnologías, los cuales son comparados entre sí para obtener cuál es la tecnología que mejor se resuelve los problemas presentados.

El tercer capítulo recopila los requerimientos que se encontraron en el proyecto a partir del estudio de los diferentes sistemas de transporte público, así mismo se presenta la arquitectura que se utilizará a lo largo del proyecto junto con la experiencia de usuario. El capítulo finaliza proponiendo políticas de diseño y documentación que deberían seguirse en los proyectos futuros y que actualmente se utilizan ampliamente en la industria.

En el cuarto capítulo se hace un estudio de la estructura básica de la base de datos necesaria para este proyecto. utilizando herramientas de modelado de clases, se obtienen esquemas que facilitan el entendimiento del estándar GTFS y de la estructura esperada en la base de datos, esto permite el poder replicar en otras tecnologías de bases de datos una estructura que sea compatible con el sistema.

También se da una descripción detallada de cada uno de los campos que componen las tablas de las bases de datos y el porqué se encuentran dentro del prototipo, al igual que capturas de pantalla que muestran a la base de datos trabajando junto con el servidor. Todo esto finaliza con el código SQL, el cual se encuentra en el apéndice B, que fue utilizado en este proyecto.

En el quinto capítulo se describen y analizan los protocolos y estándares de comunicaciones que se utilizan ampliamente dentro de la industria y la investigación y que serán utilizados en este proyecto. Realizando un análisis a los RFC de los protocolos, se proponen técnicas de utilización, así como buenas prácticas para tener un sistema que maneje las mejores prácticas internacionales.

De igual manera, se estudian, analizan y comparan los diferentes formatos representación de recursos, los diferentes meta datos que se pueden incluir en los paquetes y los diferentes patrones de URI que se utilizan en la industria y que se utilizarán dentro del proyecto. Estos permiten obtener un sistema que siga los estándares internacionales de diseño, así como la reutilización de paquetes de terceros que se basan en estos estándares.

Este capítulo es finalizado con una serie de propuestas para la API en términos de seguridad y de expansión a futuro. Estas se dan para dar una pequeña orientación acerca de lo que podría ser el siguiente paso para este proyecto.

Capítulo 1

Análisis del problema

Comprender los problemas que actualmente tienen los sistemas de transporte público en la Zona Metropolitana del Valle de México permitirá el diseñar un trabajo de tesis más acorde a las necesidades actuales. Es por ello que en éste capítulo se intentará estudiar algunos de los problemas existentes que tienen relevancia para el trabajo de tesis.

1.1. Descripción del problema

1.1.1. Sistemas de Transporte Público en la zona metropolitana del valle de México

La gran cantidad de personas que requieren desplazarse diariamente desde su casa hasta su trabajo han generado una gran demanda de transporte, en donde el 85 % del espacio vial es ocupado por automóviles.

Un estudio de la problemática de movilidad de la zona metropolitana del valle de México [3] demuestra que la velocidad de transporte privado promedio dentro de la zona metropolitana es de 14 [km/h], mientras que la velocidad promedio del transporte público de superficie de es 5 a 12 [km/h]. Esta desigualdad en las velocidades da nacimiento a cuellos de botella dentro de la zona metropolitana, lo que se traduce en mayor contaminación ambiental, personas con retrasos en sus agendas y en una mala calidad de los sistemas de transporte en general.

En el tabla 1.1 se muestran el porcentaje de usuarios que trasladan cada uno de los sistemas de transporte público en la zona metropolitana. Éste tabla representa la proporción de los viajes dentro de la zona metropolitana cuando sólo se toma en cuanta el transporte público:[1]:

[1] Estimación elaborada a partir del Plan Integral de Transporte de Vialidad 2007-2012

Tipo de Transporte	Porcentaje que representa
Transporte Colectivo[2]	65 %
Taxi	17 %
Metro	8 %
Suburbano	7 %
RTP	2 %
Trolebuses	1 %
Metrobús	0.5 %

Tabla 1.1: Proporción de representación de los sistemas de transporte público

En el Distrito Federal el servicio de transporte colectivo proporciona 9.6 millones de viajes diarios, que representa el 60.16 % de los viajes totales [2]. Esto se realiza a través de un parque de 30,170 unidades de las cuales 20,000 son microbuses. Sin embargo, cerca del 80 % de los vehículos se encuentran fuera de norma y han cumplido más de 10 años, plazo que se establece como el límite de su vida útil.

Muy por detrás quedan servicios prestados por sistemas como el Trolebús y el Metrobús con un 1 y 0.5 % respectivamente.

Actualmente existe una automatización de los sistemas de transporte por parte de la SETRAVI[3] que permitirá crear bancos de información geográficos que actualmente no existen. Estos serán utilizados en el estudio y desarrollo de futuros sistemas de transporte que sean más eficientes, baratos, amigables con el medio ambiente y respetuosos de las normas. Finalmente se espera que aumente la calidad, certidumbre y seguridad de los propietarios de vehículos particulares al poder revisar el comportamiento de los conductores de transporte.

1.1.2. Problemas que afectan los sistema de transporte

Debido a que no son un sistema aislado de las demás actividades humanas, existen factores importantes que afectan actualmente las vialidades de la zona metropolitana generando que el tráfico sea descontinuo, aumentando la contaminación ambiental y auditiva y al mismo tiempo produciendo problemas de salud debido al estrés. Este tipo de problemas se pueden catalogar como intrínsecos y extrínsecos dependiendo de su origen.

1.1.2.1. Problemas Intrínsecos

1. **Competencia entre transportes por los pasajeros:** Esto no afecta de igual manera a todos los sistemas, en realidad se podría establecer que sólo afecta a los sistemas de

[2]El transporte colectivo contempla los servicios prestados en vagonetas, combi, microbuses y en menor medida autobuses

[3]Secretaría de Transporte y Vialidad

transporte colectivo. El problema es debido a que los conductores subsisten de sus ingresos diarios, obligándolos a transportar la mayor cantidad de pasajeros posibles para obtener la mayor ganancia posible.

2. **Falta de seguridad en el transporte:** Los sistemas de transporte de la ciudad ofrecen una cantidad de seguridad mínima a sus usuarios. Desde la falta de una regulación estricta de la velocidad máxima a la que el conductor debe conducir, así como en la cantidad máxima de personas que pueden ser transportadas al mismo tiempo, hasta la falta de vigilancia por parte de policias en algunos sistemas o seguros de viaje en los transportes como Microbuses.

3. **Falta de regulación en los tiempos del recorrido:** A cada conductor se le asigna un tiempo de partida y llegada para su recorrido que no está regulado y mucho menos estudiado para saber si es el lapso óptimo para atender a la mayor cantidad de personas con el menor número de vehículos sin ponerlas en riesgo, sin mencionar que no se toma en cuenta el hecho de que el tráfico es un sistema dinámico y que los puntos óptimos del sistema están cambiando constantemente. En la mayoría de las ocasiones este límite en los lapsos puede obligar a algunos conductores a aumentar su velocidad por arriba de límites seguros poniendo en peligro a los pasajeros y al mismo tiempo creando tráfico a velocidades variables a los largo de todos sus elementos, lo que conlleva a cuellos de botella como se mencionó anteriormente.

4. **Falta de regulación de paradas:** Esto prácticamente sólo afecta a los sistemas de transporte colectivo donde las paradas son controladas principalmente por los propios pasajeros, lo que puede significar una distancia relativamente corta entre éstas, deteniendo el flujo de tráfico en múltiples ocasiones creando más cuellos de botella.

1.1.2.2. Problemas Extrínsecos

1. **Aumento del parque vehicular en la zona metropolitana:** El crecimiento de la población, la reducción en los precios de los automóviles y la facilidad de acceso a créditos con bajas tasas de interés han creado un aumento del número de vehículos particulares dentro de la ciudad, saturado las vías y reducido las opciones de viaje a lo largo de la ciudad.

   Todo esto ha hecho que los sistemas de transporte público tengan menor número de rutas alternativas para poder ir entre dos puntos.

2. **Reducción en el número de vías accesibles:** Uno de los mayores problemas que tiene gran parte del Distrito Federal es su subsuelo, el cual hace difícil mantener las calles en estado correcto. El hundimiento natural de la ciudad, así como la necesidad de cerrar zonas para reparación debido a la pobre infraestructura y planificación con la que fueron

construidas, reduce aún más las rutas alternativas que se pueden tomar y aumenta el tráfico dentro de vías alternativas.

3. **Negocios ambulantes:** Existen zonas de la zona metropolitana, como el caso de la Merced y zonas del centro histórico, en donde el número de vías que se pueden transitar se reduce debido a la invasión existente de negocios ambulantes y personas que son obligadas a caminar por la calle por la falta de espacio en las banquetas. Una pésima regulación, así como una pésima planificación por parte del gobierno, han generado que en los últimos años el problema sólo empeore más.

4. **Concentración de las actividades económicas en el núcleo urbano:** Uno de los graves problemas que conlleva una mala planificación urbana es la concentración de puntos de comercio y una distribución de las zonas residenciales en zonas externas o periféricas a estos puntos de comercio. La posición de los primeros genera una gran cantidad de tráfico provenientes de los habitantes de las zonas residenciales que necesitan trabajar o comerciar.

5. **Topología de la zona metropolitana:** La zona metropolitana ha crecido durante muchos años en una estructura desorganizada si es comparada con ciudades como Nueva York, en donde todos los bloques se encuentran alineados y en cuadrícula, por lo que las calles no siempre representan el camino más corto o el más óptimo entre un punto A y un punto B creando recorridos más largos de lo necesario. Así mismo, la intersección de vialidades importantes crea puntos potenciales donde pueden aparecer cuellos de botella.

6. **Inmigración:** El crecimiento de la zona metropolitana del valle de México está generando una nuevas fuentes de trabajo y oportunidades de negocios que atraen personas de otras partes de la república, aumentando la densidad de población y el número de personas que necesitan transporte.

Existen otros tipos de problemas que pueden llevar a reducir la eficiencia de los transportes públicos. Actualmente existen muchas teorías acerca de cómo se pueden optimizar el sistema en general, como el mantener la distancia entre los vehículos constante. Sin embargo se ha demostrado que esto no funciona, por lo que se han creado nuevas soluciones que tratan de resolver estos problemas utilizando algoritmos bio-inspirados como “antiferomonas”. [11]

Para este trabajo de tesis se han tomado en cuenta sólo los problemas intrínsecos y se diseña el sistema para que se pueda adaptar a futuras soluciones a los problemas extrínsecos, reduciendo la complejidad de la solución y permitiendo adaptarse a problemas no anticipados.

1.2. Dominio de la solución

En este trabajo de tesis se presenta el estudio y análisis de diferentes tecnologías necesarias para la construcción de un sistema que permita proponer un nuevo sistema de monitoreo vehicular.

1.2.1. Monitoreo Vehicular

Los sistemas de monitoreo que se utilizan en algunos de los transportes de la zona metropolitana, actualmente se encuentran muy segmentados y especializados en el sistema que vigilan, en donde cada uno de los diferentes sistemas tiene servidores y administraciones independientes, lo que no permite comunicación entre ellos ni con sus usuario. Sin mencionar la falta de éstos para transportes críticos como son taxis y transporte colectivo. Estos últimos utilizan sistemas basados en tarjetas por lo que no puede ser accedida en tiempo real por las autoridades que los regulan.

1. ***Sistemas de localización automática de Vehículos:*** Son sistemas automatizados de seguimiento que utilizan sistemas de posicionamiento, como el sistema de posicionamiento global (GPS[4]), y sistemas de referencia en el plano de tierra.

   Han sido utilizados por más de 20 años en vehículos de emergencia, así como en el manejo de flotas de camiones o en la recolección de información de zonas dentro de ciudades. En el pasado, estos sistemas eran diseñados utilizando puntos de señalización fijos, en donde los terminales móviles determinaban su posición utilizando su posicionamiento relativo con respecto a la posición exacta de los puntos de señalización la cual era conocida. La primera demostración de un sistema AVL usando esta tecnología fue en Chicago en los años 60.

   Para inicios de los años 90's, la tecnología de puntos de señalización comenzó a ser sustituida con sistemas GPS en parte gracias a su implementación como una tecnología de navegación, la creciente capacidad de los sistemas electrónicos y la disminución en el precio de los receptores hasta convertirse en la actualidad en la tecnología de navegación más común en este tipo de sistemas. En los últimos años la precisión de los sistemas GPS ha aumentado significativamente permitiendo actualmente tener precisión del orden de metros con pocos satélites.Existe un gran potencial en los sistemas AVL para mejorar los sistemas de transporte en una zona y modificar la cultura en el transporte público permitiendo pasar de un sistema reactivo a un sistema monitorizado que permita prevenir problemas y resolver crisis antes de que estas ocurran. A pesar de ser más baratos y precisos que sus antecesores, los sistemas GPS tienen algunas desventajas:

   a) Debido a la banda de frecuencias en la que operan y la potencia de recepción de la señal no pueden ser utilizados en zonas cubiertas como túneles o zonas con techo. Inclusive, debido a la naturaleza de las señales electromagnéticas cubiertas especiales como árboles modificarían la señal obtenida.

      Esto genera un problema para transportes como el Metro, en donde no existe recepción, haciendo necesario otro tipo de sistema.

[4]GPS: Global Position Systems

b) Los sistemas de GPS requieren de un número mínimo de satélites en linea de vista con el receptor para poder funcionar bien, el cual está dado por la precisión que se requiere en los datos.

c) Las señales sufren efectos físicos propios de las ondas electromagnéticas como son reflexión, difracción y atenuación creando errores en el procesamiento de las señales.

2. ***Sistemas de almacenamiento Geográfico:*** Estos sistemas permiten guardar coordenadas geográficas, en forma de texto o en forma binaria, dentro de una base de datos con campos especializados. Utilizando estas bases de datos geográficas, es posible crear un banco de datos donde se guarde la posición del vehículo, obtenida a través de un GPS u otro sistema de localización automática de vehículos, a lo largo del tiempo. Toda esta información puede ser accedida y analizada después para realizar estudios y mejoras en los sistemas viales de una zona geográfica establecida puede permitirían conocer si existen cuellos de botella en una ciudad, mejorar la planificación y diseño de nuevas vías y analizar posibles rutas alternas entre dos puntos permitiendo disminuir el número de vehículos en una sola vía.

1.3. Posibles aplicaciones del sistema a la administración del transporte

La administración del transporte como se entiende en este trabajo, se refiere a los procesos y servicios que se realizan dentro de las secretarias y empresas que actualmente están involucrados en este tema. Sin embargo, no todas las aplicaciones que aquí se presentan son aplicables totalmente a todos los sistemas. Un ejemplo es la asignación de nuevas rutas o diseño de rutas temporales que no puede ser aplicado en el Metro.

Entre las actividades que se pueden crear a partir de este trabajo de tesis se encuentran:

1. **Diseño de rutas:** Actualmente las rutas son asignadas por una colaboración entre la Secretaria de Comunicaciones y Transportes (SCT) y la SETRAVI. Estas son asignadas utilizando concesiones como se establece en [1]. Es obligación de la Secretaria el hacer los estudios necesarios para sustentar la necesidad de otorgar nuevas concesiones y también de *"Redistribuir, modificar y adecuar los itinerarios y rutas de acuerdo con las necesidades y las condiciones impuestas por la planificación del transporte"*.

   Mediante un sistema computacional es posible virtualizar las rutas basadas en esta información en tiempo real y diseñar rutas alternativas que permitan distribuir el tráfico en la ciudad de una manera más óptima. Esto generaría una gran cantidad de beneficios al transporte, la ciudad y el medio ambiente como son:

   a) Reducción en los tiempos de recorrido mediante recorridos alternos por vías menos congestionadas

b) Rediseño de rutas utilizando métodos de optimización utilizando información de la base de datos.

c) Creación de nuevas rutas de transporte público a partir del análisis de los datos guardados en la base de datos.

d) Reducción en el número de vehículos dentro de una vía, reduciendo al mismo tiempo el tiempo de duración de las congestiones.

2. **Control de puntos extremos:** Es posible automatizar el proceso de chequeo que deben de hacer los transportes cuando estos llegan a sus destinos. Esto permitiría saber el tiempo exacto que existió en el recorrido y analizar si la ruta del punto A al punto B es la mejor. Entre la información que se recopilaría está: quién ha llegado, en dónde, a qué hora e inclusive qué diferencia existe entre la hora de llegada real y la hora de llegada teórica o estimada permitiendo:

   a) Crear modelos más realistas del tiempo promedio de recorrido, además de poder analizar las diferencias en los tiempos de trayectos a diferentes intervalos del día.

   b) Calcular indirectamente en qué momento del día existe un mayor consumo de combustible y por ende tener un análisis de costos mucho más realista.

   c) Ayudaría a los vehículos a cumplir con el tiempo del trayecto eliminando la necesidad de ir a velocidades que pueden ser peligrosas para los pasajeros.

3. **Manejo de Turnos de Salida:** Cada uno de los sistemas de transporte público está compuesto por varios vehículos que salen en un orden fijo con intervalos de tiempo establecidos dependiendo de la cantidad de pasajeros que deben transportarse. Durante las horas con más pasajeros el volumen de vehículos en circulación es mayor dejando una menor distancia entre ellos. Sin embargo, esto no es siempre la mejor opción, por ejemplo, como se explica en [3], al aumentar la velocidad de 8 [km/h] a 20 [km/h] es posible atender la misma demanda de usuarios con un 60 % menos de vehículos.

   Los turnos de salida puede diseñarse utilizando algoritmos de optimización que podrían calcular la cantidad real de vehículos, la distancia mínima entre ellos o la velocidad a la que deberían moverse basándose en la información que está siendo recopilada por los otros vehículos dentro de las vías en estos momentos, permitiendo optimizaría recursos y evitar la saturación de las vías por un excesivo número de vehículos lanzados. Algunas de la posibilidades de un sistema de esta naturaleza son:

   a) Analizar las estrategias de salida de los vehículos y su eficiencia

   b) Elaborar cronogramas automáticos a partir de información real, en lugar de estimación

   c) Automatizar de la gestión de flotas y personal

4. **Identificación de vehículos y conductores:** Utilizando la información de las concesiones del gobierno así como las empresas que manejan los transportes públicos, es posible construir un sistema que identifique de manera única los vehículos registrados en el gobierno, así como los vehículos que están ofreciendo servicio en el momento. Esto ayudaría a regular a los servicios y reduciría todas las irregularidades que actualmente existen dentro de algunos de estos sistemas. Otras ventajas serían:

   a) Mantener un registro constante de las personas que se encuentra a cargo de cada vehículo en todo momento.

   b) Ayudar en la seguridad dentro de los vehículos al poder responsabilizar a alguien en caso de un accidente.

5. Automatización de Reportes: Permitiría respuestas más rápidas a sucesos inesperados y por ende reducción de los costos al poder responder a éstos lo más pronto posible. Otras ventajas son:

   a) Conseguir información en tiempo real sobre la respuesta de diferentes estrategias dentro del sistema

   b) Ayudar a mantener una alimentación constante de información ayudando a mejorar constantemente el sistema

   c) Reducir la cantidad de personal necesario para administrar el sistema

Es posible la existencia de una mayor cantidad de aplicaciones que podría tener un sistema de monitoreo vehicular. Los presentados anteriormente son los más evidentes y directos que el sistema podría tener.

1.4. Requerimientos generales del proyecto

A lo largo del desarrollo de esta investigación se busca que cada una de las tecnologías que conforman el prototipo y el trabajo de tesis en su conjunto, cumplan lo mejor posible los siguientes puntos:

1. No incumplan con ningún reglamento o ley
2. Sigan una arquitectura modular que permita darle mantenimiento más fácilmente
3. Los módulos que componga el sistema deben ser independientes entre sí
4. Utilicen tecnologías de código fuente abierto con licencias que permitan la modificación del código a futuro
5. Los paquetes y códigos utilizados deben estar actualizados y exista algún equipo realizando el mantenimiento actualmente

6. Cumplan los estándares internacionales en desarrollo de software, así como en el manejo y almacenamiento de la información

7. Permitan automatizar la mayor cantidad de procesos posibles, sin aumentar la complejidad de estos.

8. Posibiliten la variación de los costos para poder adaptarlos dependiendo de las necesidades

9. Permitan su implementación y adaptación a cualquier otro tipo de sistema que pueda nacer en el futuro

Los requerimientos generales del proyecto, así como los problemas estudiados en éste capítulo, serán utilizados en el siguiente capítulo para realizar una selección de las tecnologias que se utilizarán en éste trabajo de tesis.

Capítulo 2

Estudio del arte de las tecnologías

En este capítulo se estudiarán y compararán diferentes tecnologías computacionales que podrían ser utilizadas en el prototipo. Las tecnologías de desarrollo que se estudien deben de tener como base las siguientes características:

1. **Robustez:** La plataforma debe de mantenerse estable no importando la cantidad de información que sea introducida dentro del sistema

2. **Confiabilidad:** Todas las tecnologías que se utilicen deben de haber sido testeadas en mercado, esto permite saber que soportan aplicaciones de gran intensidad

3. **Escalabilidad:** Las tecnologías deben permitir aumentar la cantidad de sistemas que se pueden controlar, con esto se logra eliminar la necesidad de comenzar el sistema desde cero.

Aunque es complicado encontrar tecnología que cumplan las tres características al mismo tiempo, ya sea porque los nuevos sistemas que permiten una fácil escalabilidad son experimentales o porque un sistema demasiado robusto es poco adaptable a cambios, se intentarán encontrar aquellas que cumplan lo mejor posible con estas características.

2.1. Selección del lenguaje de programación

Para este trabajo de tesis se analizaron varios lenguajes y se escogieron tres lenguajes que cumplen con las características mencionadas anteriormente, a continuación se muestra un resumen de cada uno y una pequeña comparativa entre ellos.

2.1.1. El lenguaje de programación Python

Python es clasificado como uno de los lenguajes de programación más sencillos de utilizar gracias a su sintaxis de programación sencilla de leer y escribir, al ser un lenguaje de alto nivel es más parecido al lenguaje humano que al lenguaje máquina y es un lenguaje multiparadigma.

Al mismo tiempo también es un lenguaje muy expresivo, lo que significa que con pocas lineas de código se obtiene un una aplicación cuyo equivalente en lenguajes compilados como C++ o Java llevaría cientos de lineas.

Python también es un lenguaje multiplataforma, por lo que en general el mismo código es interpretado correctamente en Windows o Sistemas derivados de Unix como Linux, BSD o Mac OSX sin necesidad de reconstruir o recompilar. Sin embargo, también es posible construir aplicaciones para plataformas específicas, pero esto raramente es necesario.

Una de las grandes ventajas que tiene este lenguaje es su librería estándar, junto con miles de librerías, frameworks[1] y paquetes externos construidos por la comunidad que permiten entre otras cosas:

1. Descargar archivos desde internet
2. Concurrencia[2]
3. Programación distribuida
4. Realizar procesamiento de datos
5. Compresión de datos y archivos
6. Lectura y escritura de archivos
7. Diseño y administración de servidores web
8. Sistemas de Información Geográfica

Las librerías, frameworks y paquetes externos pueden verse como complementos a la librería estándar que proveen nuevas funcionalidades. Entre los más famosos se tienen SimPy, NumPy, Twisted, Pytest, Gunicorn, Django, entre otras. [50]

Otra de las ventajas que tiene este lenguaje es su dinamismo en los paradigmas de programación permitiendo la programación por procedimientos, programación orientada a objetos, programación orientada a eventos, programación basada en flujos, etc.

Finalmente es necesario mencionar que uno de los grandes inconvenientes de Python hasta ahora ha sido que los sistemas interpretados son más lentos que los sistemas compilados. Esto se ha tratado de resolver con herramientas como Cython[3] o Pypy[4] así como con la integración

[1]Frameworks: La palabra inglesa "framework"(marco de trabajo) define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

[2]Concurrencia: En computación, la concurrencia es la propiedad de los sistemas que permiten que múltiples procesos sean ejecutados al mismo tiempo, y que potencialmente puedan interactuar entre sí

[3]Cython: Herramienta que permite crear extensiones en C para el lenguaje de una manera sencilla

[4]Pypy: Interprete que permite hace optimización en tiempo real, logrando velocidades cercanas a la de un lenguaje compilado

de otros lenguajes como son Java (Jython[5]) y .NET (IronPython[6]) que permite velocidades parecidas a los lenguajes de compilados.

2.1.1.1. Características de Python

Algunas características del lenguaje Python son:

1. Sencillo de programar
2. Compatible con otros lenguajes de programación
3. Permite una gran cantidad de paradigmas de programación
4. Existen módulos de código fuente abierto que permiten resolver muchos problemas
5. Extensible
6. Velocidad de respuesta alta
7. Un gran soporte de bases de datos a través de paquetes externos
8. Posibilidad de trabajar desde computación científica hasta aplicaciones web

2.1.2. El lenguaje de programación Java

Java es uno de los lenguajes más extendidos del mundo. Está basado en un sistema de compilado intermedio (bytecode) que es interpretado en una Máquina Virtual (JVM[7]), la cual es posible instalar en cualquier sistema operativo.

La máquina virtual permite a Java ser un lenguaje independiente de la plataforma y arquitectura sobre la que se ejecuta, lo que genera que una vez compilado el código en una arquitectura de procesador es posible ejecutarlo en otras arquitecturas.

2.1.2.1. Características de Java

Algunas características importantes del lenguaje JAVA son:

1. Independencia de plataforma
2. El API del lenguaje funciona como una librería
3. Paradigma de programación Orientado a Objetos
4. Compatible con Servicios Web

[5]Jython: Integración de Python con el lenguaje Java que le permite llamar cualquier clase de Java.
[6]IronPython: Implementación del lenguaje Python que es compatible con el framework .NET y Mono
[7]JVM: Java Virtual Machine

5. Permite programación con sistemas multihilo
6. Tiene un amplio soporte de bases de datos
7. Permite técnicas como AJAX a través de servidores de aplicaciones

Java permite diseñar sistemas orientados a la web a través de tecnologías como JSP's, JSF's, y Applets.

2.1.3. El lenguajes de programación PHP

Lenguaje de propósito general ampliamente utilizado en los servicios web, gracias a su facilidad de programación y la posibilidad de embeberlo en archivos HTML.

Las siglas de este lenguaje provienen de Preprocesador de Hipertexto por lo que se puede llevar a cabo procesamiento con lenguajes de hipertexto del lado de servidor tales como HTML, XML o cualquier derivado del estándar SGML.

2.1.3.1. Características de PHP

Algunas características importantes del lenguaje PHP son:

1. Es principalmente orientado a Web
2. No es propiamente un lenguaje orientado a objetos, aunque es posible definir clases
3. Es un lenguaje con un amplio soporte de bases de datos
4. Es de código fuente abierto
5. Permite el acceso a archivos remotamente
6. Es capaz de establecer conexiones en arquitecturas cliente servidor a través de sockets
7. Permite tener seguridad embebida en el sistema
8. Es soportado por la mayoría de los servidores actualmente
9. Su configuración se realiza a través de archivos XML

Todas estas características hacen a este lenguaje ideal para aplicaciones que requieran conexiones a bases de datos y manejen una baja cantidad de información.

2.1.4. Comparación entre los lenguajes

En el tabla 2.1 se muestra una comparativa sobre los lenguajes de programación que servirá para tomar la decisión acerca de cuál utilizar a lo largo de este proyecto.

Características	Python	Java	PHP
Multiplataforma	X	X	X
Orientado a objetos	X	X	
Permite manejar otros paradigmas de programación	X		X
Disponible en servidores Web	X	X	X
Librerías externas especializadas	X	X	X
Manejo de seguridad	X	X	X
Open Source	X		X
Conexión a múltiples bases de datos	X	X	X
Web Services	X	X	
Comunidad de desarrollo	X		X
Integración con otros lenguajes	X		
Permite sistemas multihilo fácilmente	X	X	
Soporta sistemas multiconcurrentes	X	X	
Permite hilos específicos de usuario	X		
Posibilidad de programación de sistemas embebidos	X		
Servidores multihilos y con soporte de alta concurrencia	X		X
Resuelve el problema C10K	X	X	

Tabla 2.1: Comparación de los lenguajes de programación

El lenguaje de programación Python tiene, para los fines de este trabajo de tesis, las características que mejor se adaptan a los requerimientos.

Permite una adaptación rápida a la web a través de frameworks como Django, es posible manejar servidores de alto nivel como Gunicorn o Tornado e inclusive es posible integrarlo con servidores de big data como Apache Hadoop.

Sus estándares y prácticas de programación hacen muy sencilla la documentación y la posible creación de un equipo para manejar el sistema, todo esto sin perder robustez, seguridad y velocidad.

La portabilidad de este sistema, permite a las aplicaciones ejecutarse en diferentes sistemas operativos y su licencia es abierta a diferencia de Java que pertenece a la empresa Oracle.

2.1.5. Análisis del ambiente Python

El lenguaje de programación Python ha tenido una gran evolución desde su creación en 1990 por Guido van Rossum [50]. Esto ha generado un lenguaje que es estable, dinámico, principalmente orientado a objetos y multiplataforma.

Desde sus inicios python fue pensado como un lenguaje de fácil uso y aprendizaje, sin embargo, esas características terminaron dando un lenguaje que permite alta productiva en

todas las fases del ciclo de vida del software:

1. Análisis
2. Diseño
3. Generación de prototipo
4. Creación del código
5. Testeo
6. Depuración
7. Mantenimiento

Generando un manera elegante, simple y práctica de ver el diseño de software en todos los niveles.

Una vez que un lenguaje de programación ofrece una buena manera de expresar una idea, agregar otra manera solo tiene beneficios modestos, mientras que la complejidad del lenguaje puede crecer no linealmente conforme se aumentan las características. Un lenguaje complejo es difícil de entender, difícil de implementar sin errores y difícil de mantener. Es por ello que python es un lenguaje utilizado en grandes proyectos de código, en donde es necesario entender y mantener código escrito por otros.

Python es simple, pero no simplista. En este lenguaje existe la filosofía de que un lenguaje que se comporta de un manera en ciertos contextos se debería manejar igual en todos los contextos. Al mismo tiempo, se respeta el que un lenguaje no debería tener ”convenientes“ atajos, casos especiales, excepciones puntuales, distinciones sutiles o misteriosas optimizaciones ocultas. Lo que se traduce en que un dado un problema resuelto de manera similar por dos programadores distintos, los códigos en Python la mayoría de las veces sean iguales.

Python es un lenguaje de muy alto nivel. Esto significa que utiliza una mayor abstracción a la que utilizan lenguajes como C, C++ o Fortran, los cuales son considerados lenguajes de alto nivel. También es más simple, rápido en procesamiento y mucho más regular que la mayoría de los otros lenguajes de alto nivel.

En los lenguajes compilados la generación de lenguaje máquina y las posibles optimizaciones del código en tiempo de compilación permiten una ejecución más rápida que la que se podría alcanzar código de cualquier lenguaje interpretado. Sin embargo, en la mayoría de los casos la velocidad que se puede obtener en Python es suficiente. Si aún así es necesario aumentar la velocidad proyectos como Pypy y Cython que permiten una interpretación más rápida del propio código.

Por ahora Pypy se presenta como una oportunidad muy interesante al permitir un aumento en la velocidad de programa sin necesidad de modificar el código fuente. En la figura 2.1 se puede ver una comparativa de las diferentes implementaciones interpretadas con Pypy y su

velocidad de ejecución para ciertos paquetes normalizado con respecto al intérprete más usado de Python:[8]

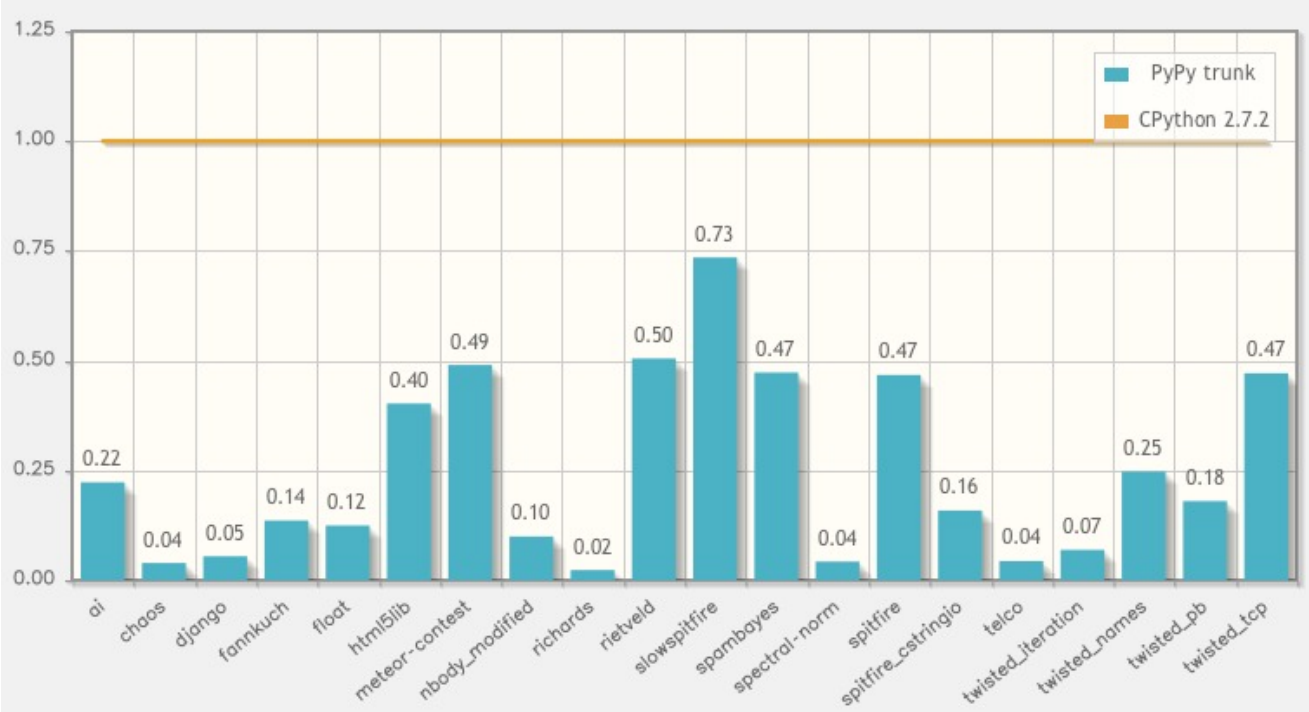

Figura 2.1: Representación del tiempo de interpretación de Pypy

La librería estándar y los paquetes externos son parte integral del propio lenguaje. La primera provee módulos completamente diseñados en Python con una implementación de alto nivel y que son mantenidos por los desarrolladores del lenguaje. Ésta puede llevar a cabo tareas como representación de datos y cadenas, procesamiento de textos, interacción con con sistemas operativos y sistemas de archivos, programación web básica, programación científica y programación de alto desempeño. El principal problema que presenta son sus actualizaciones, las cuales sólo son liberadas cada vez que una nueva versión del lenguaje sale.

Para eliminar el problema de la lentitud de desarrollo de la librería estándar, así como aumentar la capacidad del lenguaje nacen los paquetes externos que pueden estar escritos ya sea en Python o en C/C++. Aquellos escritos en los lenguajes C/C++ pueden ser conectados con herramientas como SWIG[9].

Finalmente, Python tiene un rico ecosistema dedicado a los servicios Web, entre los que se destacan:

1. **Django:** Es un Framework web de alto nivel escrito en python que permite un rápido desarrollo de aplicaciones web. [22]

2. **Tornado:** Es un Framework web escrito en python con una librería asíncrona desarrollado

[8] Gráfico tomado de speed.pypy.org

[9] SWIG es una herramienta multipropósito que permite crear interfaces entre diferentes lenguajes y el lenguaje C

originalmente por FriendFeed que utiliza entradas no bloqueadas que le permiten superar el problema C10K [23]

3. **Twisted:** Es un motor de redes de eventos escrito en python bajo la licencia MIT el cual soporta una gran cantidad de protocolos y sockets programado utilizado el paradigma basado en eventos.

Otros lenguajes que se analizaron fueron C/C++ que debido a la dificultad de desarrollo y a su sintaxis fueron descartados para un proyecto piloto ya que lo único que ofrecen adicional a lenguajes como Python es velocidad de ejecución la cual no es muy necesaria.

Go es otro lenguaje analizado, está siendo desarrollado por ingenieros Google y a sido liberado bajo la licencia MIT. Entre sus ventajas está el hecho de que combina una sintaxis de alto nivel como Python con la compilación a código máquina como C++, es por ello que sus creadores dicen que es una fusión de estos dos lenguajes. Sin embargo, éste lenguaje está en una fase muy temprana en el momento que este estudio se llevó acabo y por ello se tuvo que descartar, aunque es un lenguaje que se debería de considerar para el sistema en un futuro.

Una vez estudiado y seleccionado el lenguaje de programación, se puede pasar a estudiar y comparar sistemas de seguimiento que se pueden utilizar dentro del prototipo.

2.2. Estudio de los sistemas de seguimiento

Un sistema de seguimiento se puede definir como todo aquel dispositivo electrónico que es capaz de saber su posición aproximada o exacta en todo momento y enviarla a diferentes sistemas como servidores o interfaces de usuario.

Al ser un dispositivo que interactúa con el usuario y al que los administradores pocas veces tendrán acceso, debe ser elegido tal que su facilidad de uso, construcción, instalación y mantenimiento no requiera de conocimientos muy avanzados por parte de los usuarios y que al mismo tiempo admita otros medios de transporte. Cuando se requiera añadir un nuevo tipo de transporte al sistema, sólo debería ser necesario instalar otro de estos sistemas y comenzar a operar.

Aunque un dispositivo de estas características es difícil de conseguir, actualmente existen dispositivos que se permiten la mayoría de las características mencionadas.

Resumiendo, las características que se buscarían son:

1. **Fácil de instalar:** El agregar otro dispositivo a la red debería ser sencillo y rápido permitiendo cubrir la demanda tan pronto como ésta aparezca. Tampoco debe de complicar el sistema de recepción o el medio de transmisión para su funcionamiento.

2. **Independiente del fabricante:** Debido a que no todas las necesidades son iguales, es posible que se requieran diferentes tipos de dispositivos o marcas. Esto no debe de

ser un impedimento para la comunicación con el servidor, lo que obliga que se utilicen principalmente estándares internacionales.

3. **Robusto:** Debe de haber sido testeado ya en aplicaciones industriales y debe de soportar vibraciones producidas por el vehículo y el camino.

4. **Fácil de conseguir:** Los componentes que compongan el dispositivo o el propio dispositivo deben de ser sencillos de conseguir dentro de México. Con esto también se asegura que habrá ayuda técnica y garantía cuando sea adquirido.

5. **Fácil de operar:** Los usuarios deben tener la capacidad de instalar un repuesto en caso de fallo. Al mismo tiempo, debe ser sencillo de utilizar de manera que no se requieran cursos de capacitación que eleven el costo innecesariamente.

6. **Escalable:** Debe de permitir añadirle funcionalidades extra que no se contemplen en este trabajo pero que sea necesarias en el futuro. Si es posible, se dará preferencia a aquellos que permitan esto con el menor número de requerimientos.

7. **Fácil de mantener:** Debe de ser sencillo de mantener por diferentes personas especializadas en el tema. No debe de ser necesario reconstruirlo desde cero cada vez que se encuentra un error.

8. **Consumir poca energía:** Debido a que este irá conectado a los vehículos y muy probablemente a la batería es importante contemplar un consumo pequeño de corriente para no sobrecargar el sistema eléctrico del vehículo.

9. **Cumpla con estándares internacionales:** No debería utilizar estándares o protocolos privados. También debería permitir la modificación de sus componentes o partes, ya sea a través de hardware o software, para adaptarlo a las necesidades que surjan.

Aunque existen muchas plataformas de seguimiento, muy pocas cumplen con más de la mitad de las características mencionadas arriba. De éstas últimas existen dos plataformas que cumplen con la mayoría. Un resumen de ellas es presentado a continuación:

2.2.1. Celulares inteligentes

La evolución e introducción de los celulares en la vida diaria a hecho que lo precios disminuyan mientras que la capacidad de estos ha crecido enormemente.

Actualmente los teléfonos celulares tienen una gran cantidad de sensores que es posible explotar para diferentes aplicaciones como lo demuestra la gran variedad de aplicaciones que existen.La aplicación puede utilizar desde medidas físicas directamente obtenidas por el celular con sensores de campo magnético, acelerómetros, entre otros. También puede utilizar hardware que interactúe con el usuario y realizar procesamiento sobre las señales como la cámara y el

micrófono. También puede realizar procesamiento en servidores para sistemas más intensivos como reconocimiento de voz. Al mismo tiempo, los celulares inteligentes tienen la ventaja de que vienen preconfigurados desde fábrica para trabajar instantáneamente con redes GSM, 3G, 4G, Wifi, Bluethoot sin necesidad de diseñarlos, construirlos o configurarlos. La utilización de éstos módulos es tan sencillo como mandar a pedir su activación al sistema operativo.

La escalabilidad de estos sistemas también es muy buena, si es necesario un nuevo dispositivo dentro de la red sólo con descargar la aplicación al nuevo celular lo dejará listo para funcionar. Finalmente una de las mayores ventajas que tienen los sistemas celulares, es la posibilidad de obtener información de su posición de diferentes fuentes como redes celulares o antenas wifi, lo cual ayuda en ciudades en donde puede haber muchas distorsiones o una recepción correcta de las señales GPS.

2.2.1.1. Características de los celulares inteligentes

1. Pueden comunicarse a través de diferentes protocolos con diferentes tipos de redes
2. Su sistema está basado en software por lo que es sencillo reparar errores
3. Funcionan con sistemas operativos lo que permite realizar procesamiento mas complejo dentro del dispositivo
4. Existen una gran cantidad de aplicaciones actualmente que realizan el seguimiento de dispositivos
5. Permiten una configuración dinámica de las redes a utilizar basado en lógica computacional
6. Permiten comunicación a través de voz con las unidades
7. Permiten enviar señales de emergencia a otros servicios (Policía, Cruz Roja, Bomberos)
8. Es posible detectar en tiempo real cuando algo va mal con el dispositivo
9. Los dispositivos cuentan con un sistema de registro en donde se guarda todo lo que sucede en el dispositivo

2.2.2. Open GPS tracker

Existen una gran cantidad de proyectos de código fuente abierto en cuanto a seguimiento se trata. El proyecto Open GPS Tracker se destaca por estar muy bien desarrollado y tener un comunidad que le da un amplio soporte.

Éste proyecto utiliza el receptor GPS A1035 que es uno de los mejores en el mercado aunque requiere una gran cantidad de potencia para poder trabajar[10] [34]. Contiene un receptor

[10]Las especificaciones del receptor establecen 50 [ma] y 3.3[V]

de ondas y un CPU que es más veloz que un microcontrolador convencional. El módulo utiliza un transistor PNP como interruptor y guarda en memoria RAM la información de la posición.

La información guardada en la memoria es transmitida por el puerto serial a 4800 [bps] utilizando un chip MAX3232.

El módulo se programa en lenguaje ensamblador o lenguaje C con un programa principal que mantenga el ciclo trabajando en un bucle, y cuatro interrupciones programadas:

1. **Timer 1 compare A:** reloj para transmisión y recepción
2. **Pin change interrupt 0:** Detecta el bit de inicio de un byte de entrada
3. **Timer 0 overflow:** Interrumpe los relojes y los códigos
4. **Watchdog timer overflow:** Inicia el procesador y lo saca del modo ”sleep“

2.2.2.1. Características del sistema Open GPS tracker

1. Los programas son convertidos a lenguaje máquina permitiendo un procesado en tiempo real de las señales
2. El sistema es completamente diseñado desde cero lo que permite un control sobre la electrónica
3. El hardware es abierto y permite que cualquier persona lo modifique a sus necesidades
4. Es posible encontrar las partes electrónicas en cualquier lugar
5. Si una parte se descompone sólo debe de cambiarse esa parte

2.2.3. Comparación entre dispositivos

En el tabla 2.2 se muestra una comparativa de los dispositivos que se tomaron en cuenta para este trabajo de tesis

Características	Celulares	Open GPS
Escalable	X	X
Funcionamiento basado en software	X	
Fácil de mantener	X	
Posibilidad de información en tiempo real	X	X
Posibilidad de procesamiento mayor con el mismo hardware	X	
Capacidad de comunicarse con diferentes redes al mismo tiempo	X	
Compatibilidad con sistemas externos (API)	X	
Comunicación a través de voz	X	

Tabla 2.2: Comparación de los dispositivos con GPS

Los sistemas celulares son por mucho uno de los mejores dispositivos que nos podemos encontrar hoy en día sin contar a los GPS comerciales como la marca Tom Tom. Además de que permiten personalizar y adaptar a nuestras necesidades a través de sólo software que puede funcionar a través de cualquier dispositivo que pueda interpretar nuestro código.

Actualmente en el mercado existe variedad de sistemas operativos que funcionan en estos dispositivos. Desde sistemas comerciales como Microsoft Windows e iOS hasta sistemas de licencia abierta como Android, Firefox OS o Ubuntu OS. Todos ellos permiten el crear fácilmente un sistema que tenga las funcionalidades que se necesitan para realizar seguimiento.

Basado en todas las ventajas que un celular puede traer al sistema se recomienda su utilización en todos los vehículos que aún no tienen un sistema de seguimiento. El sistema operativo que se puede elegir es Android debido a la facilidad de encontrar dispositivos en el mercado, así como a la gran documentación en este sistema operativo.

Tan importante como los dispositivos de seguimiento son las bases de datos, es por ello que dedicaremos la siguiente sección a estudiar diferentes propuestas que existen actualmente en el mercado.

2.3. Selección de la base de datos

Una base de datos se puede definir de una manera básica como un sistema que mantiene seguro una colección de datos. No necesariamente deben de encontrarse almacenados en forma electrónica; un catálogo o una hoja de cálculo o simplemente un archivo de texto sirven pueden funcionar como bases de datos. Dentro de este trabajo de tesis, una base de datos será definida como un contenedor electrónico ubicado dentro de un servidor que permite guardar información en forma ordenada y clasificarla dentro de un sistema de almacenamiento electrónico de archivos.

La base de datos es el corazón de los sistemas AVL, es en ellas en donde se guarda toda la información y se realizan consultas para obtener recuperarla cuando se necesite.

Una base de datos veloz es muy importante para la velocidad de respuesta de cualquier servicio web. Es la velocidad de lectura y escritura de las bases de datos, así como la posibilidad de realizar operaciones en paralelo lo que podría aumentar el tiempo que requiere un usuario para obtener información aún teniendo un servidor muy veloz.

Una base de datos debe permitir guardar la información de la manera más estructurada posible y utilizando ACID[11], con capacidad de programación de disparadores (triggers) que permitan establecer reglas de comportamiento. También debe de poder ser integrada a diferentes servidores y lenguajes de programación para que sea sencillo escalar el sistema, adaptarla a nuevas necesidades o utilizarla en aplicaciones de terceros. Finalmente, una base de datos con capacidad de manejar información geográfica es ideal para este sistema aunque no es completamente necesario al poderse utilizar otras bases de datos que sean adaptadas.

[11] ACID: Atomicity, Consistency, Isolation, Durability

Lo que hace que las bases de datos con capacidad de manejo de información geográfica sean diferentes de una base de datos de propósito general es su especialización en sistemas espaciales. Una mejor forma de analizar las razones que existen detrás de esto es comparando la cantidad de cálculos que son necesarios y el nivel al que estos son ejecutados para obtener una medida. En las bases de datos con soportes geográficos es posible obtener la medida dentro de la propia sentencia de la base de datos, por ejemplo en Postgres las instrucciones para obtener un área es:

SELECT ST_GeomFromText('POLYGON((52 218, 139 82, 262 207, 245 261, 207 267, 153 207, 125 235, 90 270, 55 244, 51 219, 52 218))') As MyPolygon;

Mientras que obtener un cálculo similar en una base de datos sin soporte para este tipo de datos, requiere una librería en el servidor que obtenga los puntos, calcule aquellos que que se encuentran en la misma área y luego lo asigne a un objeto llamado MyPolygon.

Los sistemas GIS utilizan información geométrica, que no es otra cosa más que una información en un campo binario que sistemas y procesos especializados saben como interpretar. Sin embargo, es posible tomar una base de datos general y escribir métodos que le permitan hacer los mismos procesos especializados que se esperaría de éstas bases de datos GIS.

En resumen, una base de datos especializada en sistemas espaciales tiene muchas más capacidades de entender la información que se le agrega, además de una gran cantidad de extensiones para los lenguajes SQL que permiten realizar operaciones GIS especiales, nuevos tipos de indexación que ayudan a acelerar búsquedas y varias tablas especializadas para el manejo de meta-datos relacionados con la información que es guardada.

Entre las bases de datos SQL que soportan sistemas GIS tenemos Postgres con PostGIS, MySQL y Microsoft SQL Server.

2.3.1. Base de datos Postgres

Actualmente esta es probablemente una de las soluciones para sistemas de información Geográfica más completa que existe. Sigue completamente los estándares internacionales, es utilizada alrededor del mundo por muchas empresas y proyectos de investigación, tiene un tiempo de respuesta muy bien estudiado y es soportado por cada uno de los sistemas GIS existentes.

Postgres es una base de datos que nació en 1986, desarrollada por el profesor Michael Stonebraker en la universidad de California, Berkley. En 1995, dos estudiantes del profesor Stonebraker extendieron Postgres hacia SQL, y el código fuente fue liberado en 1996. Es considerada actualmente como la base de datos de código fuente abierto más avanzada del mundo, proporcionando un gran número de características que la hacen una gran competencia para la bases de datos comerciales. [28]

2.3.1.1. Características de la base de datos Postgres

1. Tiene una implementación estándar SQL92/SQL99

2. Soporta formatos propios

3. Incorpora funciones complejas como manejo de calendarios, datos geométricos, funciones orientadas a operaciones en redes, entre otras.

4. Soporta distintos formatos de datos

5. Permite gestión de objetos relacionales a través de herencia entre las tablas

6. Soporta integridad referencial, que se utiliza para garantizar la validez de los datos de la base de datos

7. Una API soportada por la mayoría de los lenguajes de programación como Python, PHP, Java, Ruby, C++, TCL, etc.

8. Control de Concurrencia de Multi Versión (MVCC[12]), sistema que es utilizado para eliminar bloqueos innecesarios.

9. Write ahead loggin, que permite incrementar la dependencia de la base de datos al registro de cambios antes de que estos sean escritos en la base de datos, lo que hace que si la base de datos se cae, es posible recuperarla a través de un registro.

2.3.2. Base de datos MySQL

Por otra parte MYSQL, originalmente desarrollado como una base de datos simple de utilizar, a partir de la versión 3.23 incluye soporte para datos geométricos.

La documentación de la versión 3.23 [24] establece que la base de datos no sigue los estándares internacionales, en particular algunas tablas de meta-datos y muchas de las funciones geométricas tiene nombres distintos a los que el estándar establece [31] [32], por lo que es difícil dar soporte a esta versión a través de diferentes plataformas.

A partir de la versión 5.0 [25] se ha adaptado a los estándares soportando ahora una gran cantidad de operaciones GIS, por lo que ya no es necesario instalar sistemas de terceros para permitir manejo de información espacial. Esta base de datos se caracteriza por ser muy ligera y veloz, además de ser sencilla de administrar y tener una gran comunidad por detrás desarrollándola. Su único punto débil es que la licencia pertenece a la empresa Oracle y actualmente no se sabe cuál será el futuro de esta base al ser una competencia directa para la empresa.

[12]Multi Version Currency Control

2.3.2.1. Características de la base de datos MySQL

1. Utiliza herramientas GNU como Automake, Autoconf y Libtool para portabilidad
2. Su API es ampliamente soportada por una gran cantidad de lenguajes de programación como Python, C, C++, Eiffel, Java, Ruby, PHP, etc
3. Tiene un uso completo de sistemas multihilos mediantes hilos embebidos en el kernel
4. Proporciona sistemas de almacenamiento transaccional y no transaccional
5. Posee un sistema de reserva de memoria basado en hilos
6. Permite añadir otro sistema de almacenamiento muy fácilmente
7. Tablas en disco B-tree (MyISAM) muy rápidas con compresión de índice
8. Soporte completo para operadores y funciones
9. Sistema de privilegios con contraseñas
10. Soporte de grandes bases de datos
11. Las conexiones con el servidor utilizan sockets TCP/IP en cualquier plataforma.
12. Tiene comandos para optimizar, chequear y reparar tablas

2.3.3. Base de datos Microsoft SQL Server

Es una base de datos de versión ligera producto de la compañía Microsoft de utilización gratuita. Posee un regulador interno para trabajo y no permite manejar bases de datos muy grandes.

La funcionalidad GIS es una adquisición de la base que fue agregada en la versión 2008 [26]. Antes de esto, existían librerías de terceros que permitían estas funcionalidades en las versiones 2003 y 2005.

Una de las características especiales de esta base de datos es que utiliza estándares privativos de Microsoft que no son compatibles con los estándares internacionales. Esto es debido a que es un servidor basado en SQL CLR, por lo que las funciones no pueden ser accedidas de la misma manera que en las otras bases de datos.

2.3.3.1. Características de la base de datos Microsoft SQL Server

1. Compatibilidad con esquemas XML
2. Cliente para SQL Server broker
3. Aprovechamiento de los procedimientos almacenados como capa de abstracción

4. Optimizador avanzado de consultas
5. Manejo de tres niveles de seguridad de acceso al código
6. Compatibilidad con transacciones distribuidas

2.3.4. Comparación entre bases de datos

En el tabla 2.3 se ha hecho un resumen de las características que se buscan en una base de datos para poder elegir cuál es mejor para el sistema

Características	Postgres	MySQL	Microsoft SQL
Gratuito	X	X	X
Código fuente abierto	X	X	
Multihilos	X	X	X
Estándares Internacionales	X	X	
Control de Concurrencia	X	X	X
Recuperación de fallos	X	X	X
Funciones definidas por el usuario	X	X	X
Disparadores y procesamientos almacenados	X	X	X
Multi Herencia entre tablas	X		
Soportado por múltiples plataformas	X	X	
Manejo de datos geométricos y espaciales como nativos	X	X	X

Tabla 2.3: Tabla comparativo entre bases de datos

En las bases de datos no existe una gran cantidad en las características que ofrecen, en realidad las tres son buenas opciones debido a que cada una intenta en convertirse en un estándar de la industria.Aunque puedan llegar a faltar algunas características en algunas es muy posible que sean soportadas en futuras versiones por las hace perfectas candidatas para proyectos de este tipo.

Para este trabajo de tesis se escogerá Postgres debido a que es una base de datos de código fuente abierto, sigue los estándares internacionales al pie de la letra, es robusta y muy ligera, además de que permite optimización de tiempos de respuesta, y funciones definidas por el usuario para la obtención de resultados con un menor número de pasos. A pesar de que MySQL también soporta gran parte de lo que soporta Postgres, debido a su licencia, el estar a merced de Oracle y la incertidumbre que existe acerca de cual será el futuro de esta base lo que hizo que no fuese elegida.

2.3.5. Análisis del ecosistema Postgres y PostGIS

Postgres es una base de datos, libre, gratuita y de código fuente abierto con capacidad de manejar información espacial gracias a librerías especiales y con un sistema de manejo para base de datos de objetos relacionales (ORDBMS[13]).

Una base de datos de objetos relacionales es aquella que permite guardar tipos de objetos complejos en atributos en forma de tablas, lo que permite a los usuarios definir sus propios tipos de datos, nuevas funciones y operadores para manipular estos datos.

Postgres es actualmente la más avanzada base de datos de código fuente abierto. Tiene la velocidad y funcionalidad para poder competir con la más poderosas bases de datos ofrecidas por la industria y puede ser utilizado para manejar terabytes de información. Esta base datos tiene muchas funcionalidades que difícilmente se pueden encontrar en otras bases de datos:

1. **Las funciones en la base de datos así como funciones de estructura pueden ser escritas en diferentes lenguajes de programación y pueden retornar conjuntos de datos:** Pocas bases de datos de código fuente abierto y ninguna comercial hasta la fecha permite realizar este tipo de funciones. Los lenguajes más conocidos para realizar este tipo de construcciones son SQL, PL/PGSQL y C. Además de estos tres lenguajes, Pl/Perl, PL/Python, PL/TCL, PL/SH, PL/R y PL/Java también pueden ser utilizados.

   Algunas bases de datos como Microsoft SQL y DB2 permiten agregar a través de .NET funciones, aunque éstas no son escritas en el motor de la base de datos. Oracle soporta PL/SQL y Java pero con una aproximación menor a Postgres.

2. **Soporte de arreglos[14]:** En este campo Postgres, Oracle y DB2 de IBM son las únicas en las que los arreglos son datos nativos. Dentro de Postgres es posible definir cualquier atributo en una tabla para que trabaje como un arreglo de cualquier tipo de dato. Esto permite análisis de matrices o agregaciones matriciales.

3. **Herencia entre tablas:** Esta base de datos tiene una funcionalidad de herencia entre tablas que funciona de una manera muy similar a la herencia en la programación Orientada a Objetos. Esto significa que es posible tratar múltiples tablas como si fuesen una sola, lo cual da muchas posibilidades.

4. **Posibilidad de ejecutar múltiples funciones en más de una columna:** Las funcionalidades multicolumna son pocas veces explotadas correctamente aunque permiten realizar grandes cantidades de operaciones sobre grandes cantidades de datos en poco tiempo.

[13] ORDBMS: Object-Relational database Managment System

[14] Arreglos: Estructuras de datos ordenadas que permiten acceder elementos dentro de ella a través de índices numéricos

5. **Miles de funciones internas y funciones externas:** Gracias a la comunidad que soporta esta base de datos existen funciones para la mayoría de los problemas más comunes, desde la manipulación de cadenas de texto y expresiones regulares hasta minería de datos.

6. **Soporte ANSI-SQL:** Estándar que define las operaciones y el nombre de operaciones dentro de bases de datos. Permite interoperabilidad entre bases de datos y sistemas.

7. **Un sistema de planificación SQL e indexación de objetos complejos que permite optimizar operaciones SQL muy grandes:** La velocidad que se puede alcanzar es comparable con bases de datos empresariales comerciales.

8. **Habilidad de utilizar parámetros por default en funciones:** Esto es utilizado para realizar funciones que requieren uno o más parámetros para acceder, los cuales pueden obtener un parámetro por default que será pasado si la función es llamada sin ningún argumento.

9. **Soporte avanzado de transacciones :** Utiliza un sistema de control de versiones de concurrencias, el cual también es soportado por Oracle y Microsoft SQL Server, que permite retornar a puntos anteriores y guardar puntos actuales.

10. **Posibilidad de correr en cualquier arquitectura:** Postgres no depende del sistema operativo o arquitectura, lo cual permite trabajar con los mismos parámetros en diferentes sistemas.

11. **Posibilidad de incluir permisos a nivel de columna:** Postgres permite la administración de las autorizaciones de manera granular.

12. **Posibilidad de escribir funciones propias en cualquier lenguaje, incluido SQL:** Esto es particularmente útil cuando se analizan sistemas espaciales. El diseño de estas funciones es muy simple y da una gran funcionalidad que otras bases de datos no dan.

13. **Expresiones recursivas sobre tablas que permiten el manejo de sentencias recursivas:** Esta funcionalidad es encontrada en las bases de datos comerciales más grandes, lo hace muy útil para sentencias espaciales.

14. **Soporta restauraciones paralelas de tablas:** Funcionalidad introducida en Postgres 8.4 que permite a la base de datos restaurarse 4 veces más rápido que en versiones anteriores [27] lo cual es importante cuando se manejan bases de datos gigantescas.

15. **Triggers condicionales y Triggers a nivel de columna**

16. **Funciones anónimas a través del nuevo comando do:** Este comando fue introducido en la versión 9.0 y permite que lenguajes como Python, Perl, Ruby, TCl entre otros puedan crear código sin definiciones.

Es gracias a todas estas características y muchas nuevas que se están desarrollando que se ha considerado a Postgres y su versión espacial PostGIS como las bases de datos correctas para este sistema.

Una vez estudiadas algunas de las tecnologías que se utilizarán en la construcción del sistema se puede comenzar a realizar los análisis de cómo funcionará el prototipo.

Capítulo 3

Análisis del prototipo

En este capítulo se reunirán, analizarán y definirán las necesidades del sistema AVL en general, se propondrá la arquitectura y se estudiará el cómo se puede aplicar en la solución de este problema.

3.1. Requerimientos del prototipo

El primer punto que se debe revisar son los problemas más evidentes que tienen los sistemas de transporte público y las entidades reguladoras de la zona metropolitana, los procesos que pueden ser automatizados dentro del sistema y los problemas que puede enfrentar el proyecto. También se debe identificar a las personas que serán afectadas por este sistema debido a que no es un sistema aislado. Finalmente se debe proponer un esquema general del proyecto final, los módulos generales que son necesarios para trabajar en él y establecer cómo podrían interactuar con las personas y subsistemas en el día a día.

3.1.1. Estudio del producto

1. **Resumen del problema:**

   a) ***Problema:*** Actualmente no existe un método preciso y en tiempo real de obtener la posición del vehículo para algunos tipos de transporte público, mientras que en otros que sí cuentan con esta tecnología los receptores son independientes creando dificultades en la coordinación ellos y su administración.

   Al mismo tiempo las actuales rutas de transporte no han sido estudiadas para entender su efecto en la reducción de la necesidad de uso de transporte privado y en el aumento en la carga de tráfico. Los cuellos de botella tampoco son fácilmente identificables debido a que pueden ocurrir de diferentes fenómenos externos como un accidente.

   b) ***Afectados:***

1) Sistemas de transporte público
2) Pasajeros
3) Administradores de los sistemas públicos
4) Secretarías y Gobierno
5) Socios del sistema de transporte
6) Choferes de los vehículos
7) Medio Ambiente
8) Empresas privadas
9) Automóviles privados

c) ***Efectos:***

1) Algunos transportes no respetan las paradas establecidas al no tener vigilancia
2) Debido a problemas de tiempo los choferes aumentan la velocidad a la que conducen produciendo accidentes
3) El tiempo establecido para recorrer la ruta no es respetado
4) Es difícil coordinar otros sistemas de transporte cuando uno falla
5) Las vehículos gastan más gasolina al estar atrapados en el tráfico
6) Los vehículos comienzan a generar retrasos en otros vehículos al ir en tiempos distintos y velocidades distintas
7) Las unidades de transporte no mantienen la distancia y por ende la oferta teórica y la oferta real de transporte difieren.

d) ***Sistema óptimo:***

1) Permite la localización de las unidades en tiempo real
2) Automatiza una gran cantidad de tareas haciéndolas más sencillas de administrar
3) Permite diseñar calendarios y cronogramas de trabajo que se adapten a la ciudad en tiempo real
4) Permite analizar las rutas seguida y diseñar rutas alternativas que reduzcan la carga en zonas
5) Permite transmitirle toda esta información a usuarios, de manera que puedan buscar rutas alternas, reduciendo aún más el tráfico
6) Gestiona automáticamente las rutas y horarios para cada unidad
7) Maneja las penalizaciones automáticamente en las multas y castigos por no respetar las reglas
8) Genera alerta de problemas en el momento que ocurren o inclusive antes si es posible y las envía a un centro de control donde pueden manejar esto

2. **Resumen del proyecto**

   a) ***Posibles usuarios:***

      1) Administradores de los sistemas de transporte
      2) Gobierno y Secretarías
      3) Pasajeros
      4) Choferes
      5) Administradores del proyecto

   b) ***Necesidades:***

      1) Gestionar socios en el transporte, otros dispositivos AVL, puntos de control, servicios y rutas
      2) Conocer la posición geográfica de los vehículos, su velocidad, y la situación de la ruta que siguen
      3) Recibir reportes de mal comportamientos o multas
      4) Obtener la distribución real de las unidades en todo momento para estudiar la eficiencia de la oferta para cubrir la demanda de usuarios
      5) Gestionar a los usuarios y choferes
      6) Administrar en tiempo real qué sucede dentro del vehículo
      7) Obtener información crítica en tiempo real

   c) ***Propuesta:*** Se propone un sistema de monitoreo para el transporte de la zona metropolitana del Valle de México que funcione como base para un sistema posterior que cubra todas las necesidades mencionadas. El receptor estará basado en servidores y bases de datos espaciales, de código fuente abierto y de ser posible gratuitos. Éste utilizará protocolos abiertos, estándares internacionales utilizados actualmente en la industria y centros de investigación.

   d) ***Funcionalidades:***

      1) Desplegará la información de la posición y la rutas seguida por las unidades de transporte a través de una API con arquitectura REST.
      2) Recibir la información de posicionamiento de los vehículo y enviarla a una base de datos.
      3) Permitirá el acceso a toda la información recopilada para poder ser utilizada por otros proyectos.
      4) Publicar en tiempo real la información conforme esta arribe a la base de datos para que sea accedida por diferentes aplicaciones.

3. **Tipos de Usuarios:**

a) ***Administradores de los sistemas de transporte:*** Su objetivo es mantener funcionando cada uno de los sistemas que conforman el transporte público. Entre las responsabilidades que tienen están el tomar estrategias que ayuden al mejoramiento del sistema, analizar la la situación de los vehículos y hacer que los conductores sigan las rutas.

b) ***Gobierno y Secretarías:*** Su objetivo es crear las leyes y reglamentos que regirán los sistemas de transporte así como los planes de desarrollo que cada uno va a seguir. Además crean las rutas que seguirán cada uno de los transportes.

c) ***Socios privados del sistema de transporte:*** Son empresas privadas que colaboran con el gobierno en diferentes sistemas de transporte. Generalmente se encargan de mantener los vehículos en buen estado, así como administrar la infraestructura.

d) ***Despachadores:*** Son las personas que se encargan los turnos asignados para cada unidad dentro de los transportes. Entre sus obligaciones se encuentran el realizar los cronogramas que deben seguir los choferes, revisar que los vehículos salgan en intervalos definidos de tiempo y controlar el tiempo que tarda cada unidad en recorrer la ruta.

e) ***Chofer:*** Son las personas encargadas de conducir los vehículos, generalmente son varios y están divididos por horarios de trabajo. Tienen la responsabilidad de trasladar entre dos puntos a las personas de una manera segura, en un tiempo dado y a través de una ruta establecida respetando las leyes de tránsito establecidas por el Gobierno y Secretarías.

f) ***Usuarios finales:*** Son las personas que utilizan uno o más de los transportes, en donde la utilización puede ser uno sólo o una mezcla de estos. También son los que más beneficio obtendrán al mejorar los sistemas de transporte ya que les permitirá llegar a su destino con sistemas menos saturados y en menor tiempo.

Los usuarios exigen un transporte más limpio, seguro y respetuoso que los pueda llevar a un precio justo. Aunque esto parece algo muy difícil en realidad, es posible llevarlo acabo reduciendo costos a través de automatizaciones.

g) ***Administradores de sistema:*** Son aquellos que manejarán los sistemas computacionales de este proyecto una vez se esté utilizando. Entre sus obligaciones están:

1) Administrar los sistemas GPS
2) Crear, Actualizar, Mantener o Destruir las bases de datos
3) Estudiar la precisión de los datos
4) Mantener la API para acceso público
5) Asegurar la estabilidad del sistema
6) Asegurar la disponibilidad de los sistemas de captura de datos
7) Creación de interfaces para los usuarios

3.2. Módulos

3.2.1. Clasificación de los módulos

Las diferentes funcionalidades del proyecto son cubiertas a través de módulos que son independientes pero que se pueden conectar para que interactúen entre ellos. Esto reduce el tiempo necesario para darle mantenimiento al sistema y permite agregar nuevas funcionalidades a través de otros módulos sin preocuparse de generar daño.

Cada uno de los módulo se puede clasificar en tres tipos dependiendo de qué tan importante es para el funcionamiento del proyecto en general. Estos son:

1. **Crítico:** Es indispensable para que el sistema funcione
2. **Necesario:** Puede llegar a existir el sistema sin él aunque es preferible que exista
3. **Adicional:** El sistema puede existir sin él. Generalmente estos módulo permiten agregar nuevas funcionalidades.

Un módulo que actualmente es considerado *Necesario* o *Adicional* puede llegar a ser crítico para nuevas funcionalidades en el futuro. Es por ello que la clasificación que se realiza en este trabajo no debe ser considerada como definitiva.

Para este proyecto sólo se diseñarán los módulos que aquí se consideran *críticos* o *Necesarios*, mientras que los módulos *Adicionales* sólo se plantearán como propuestas.

Al mismo tiempo cada uno de los módulos tiene un grado de complejidad distinto que se relaciona con la posibilidad de encontrar soluciones ya construidas que realicen la función necesaria o la dificultad de construirlas con los actuales limitantes de las librerías, lenguajes de programación o paquetes en caso de que no existan. Esta clasificación se puede dividir en tres:

1. **Alta:** El módulo requiere de conocimientos especiales para desarrollarlo, manejarlo, repararlo y mantenerlo
2. **Media:** El módulo exige cierto entrenamiento para desarrollarlo, manejarlo y mantenerlo
3. **Baja:** El módulo es muy sencillo de manejar y no requiere mucho conocimiento para mantenerlo

En el tabla 3.1 se puede ver una lista de la complejidad e importancia de cada módulo basada en la experiencia que se obtuvo al realizar el proyecto.

Módulo	**Importancia**	**Complejidad**
API	Crítico	Alta
Sistemas de información colectiva	Adicional	Alta
Continua siguiente página		

Tabla 3.1 – continuación

Módulo	Importancia	Complejidad
Sistemas de información estática	Adicional	Alta
Información de respaldo	Necesario	Media
Sistema de gestión de dispositivos de seguimiento	Necesario	Media
Sistema de gestión de vehículos	Necesario	Baja
Sistemas gráficos	Adicional	Alta
Salidas estadísticas	Adicional	Alta
Sistemas de procesamiento manual de datos	Necesario	Media
Sistema de gestión de puntos de control	Adicional	Media
Módulos de transformación de datos	Adicional	Alta
Procesamiento	Crítico	Alta
Procesamiento de salida	Crítico	Media
Reportes	Necesario	Baja
Administración de usuarios y roles	Crítico	Baja

Tabla 3.1: Módulos que se utilizan dentro del proyecto

3.2.2. Análisis de los módulos

La arquitectura del sistema desarrollado en esta tesis, está inspirada en SOA[1] en donde la API funciona como un ESB[2] [4] de manera que todos los módulos se comunican entre sí a través de ésta y los sistemas internos se encuentran ocultos para todas las aplicaciones externas. Sin embargo, no se implementa SOA debido a que ésta forma de desarrollo es específica para sistemas distribuidos y heterogéneos, cuando es implementa mal puede complicar más que simplificar el proyecto en general.

Una diferencia importante entre el sistema propuesto en este trabajo y un sistema convencional es que cada uno de los módulos realiza una tarea específica y por ello no es necesario recrear esas funcionalidades, mientras que en los sistemas centrales es posible encontrar componentes multifuncionales que obliguen a la reconstrucción de componentes comunes.

Aunque es posible utilizar gran cantidad de librerías y paquetes para reducir el tiempo de desarrollo y el trabajo, la mayoría de las soluciones finales deben de ser adaptadas a las necesidades particulares del proyecto.

1. **Módulos de entrada:**

   a) ***API:*** Funciona como la puerta de entrada al sistema recolectando la información a través de su interfaz y enviándosela al servidor. Comúnmente la información

[1] SOA: Service Oriented Architecture
[2] ESB: Enterprise Service Bus

proviene de dispositivos externos al sistema con permisos especiales para introducir información dentro de la API.

b) ***Sistemas de información estática:*** Éste módulo ingresa información dentro de la del sistema que es registrada de manera manual como mapas existentes, información existente de rutas, información existente de vehículos, etc.

Este tipo de información generalmente es muy específica con respecto a las necesidades.

c) ***Información de respaldo:*** Éste módulo puede ser clasificado de entrada o de salida (*Entrada* cuando la información viaja del servidor de respaldo al servidor de producción y *Salida* en el sentido opuesto), se clasifica como módulo de entrada ya que es en esta situación cuando puede llegar a ocurrir un error. Toda escritura hacia el servidor de respaldo debe estar automatizada.

Debido a la gran cantidad de información que un servicio web puede producir y la importancia de la información que se guarda, es necesario contar con un sistema de respaldo independiente del sistema actual que permita trabajar con él cuando sea necesario.

d) ***Sistemas de información colectiva:*** Aunque este sistema también utiliza la API como medio de acceso al servidor y la base de datos, la fuente de la información es muy distinta. La información en este módulo viene de usuarios externos que introducen la información a través de diferentes dispositivos aprovechando la posibilidad que dan los teléfonos inteligentes y computadoras para obtener su posición de una manera sencilla. Esto permite recoger más información al mismo tiempo sin necesidad de más infraestructura dentro de los vehículos.

Es posible también que la información venga de aplicaciones "no oficiales" en donde se requiere transformar la información que llega al formato que es requerido.

2. **Módulos de salida:**

a) ***Sistemas gráficos:*** Los sistemas gráficos son los más comunes y también los más sencillos de utilizar por parte de los usuarios. Generalmente son representaciones de la información sobre mapas. Sin embargo existen otros tipos de salidas que no son tan comunes como imágenes, ambientes web, hojas de cálculo o procesadores de texto.

b) ***Salidas estadísticas:*** En éste módulo los datos de salida son productos de operaciones SQL-GIS basados en los datos de entrada y procesamiento sobre ellos. Esta información puede ser utilizada para trabajar con estadísticas y paquetes matemáticos que permitan un análisis de la información.

La mayoría de estos datos, a pesar de ser sólo números, requieren de un contexto para

poder ser analizados, es por ellos que es necesario que el sistema GIS esté envuelto en el procesamiento inicial de estos.

Un ejemplo del tipo de información que se puede obtener es por ejemplo el costo en términos de combustible de una ruta. Si se establece que el precio de la gasolina es $1.10 por litro y el transporte que se está analizando tiene un consumo de 1 litro cada 3 km en una ruta en donde la distancia entre el origen y el destino es de 16 km, es posible pedirle al sistema las diferentes rutas que existen y nos diga cuales son aquellas en las que se gasta más combustible. Aunque es posible deducir esto a partir de los mapas, las computadoras requieren de datos numéricos para poder hacer estos cálculos.

3. **Módulos de procesamiento:**

   a) ***Procesamiento manual:*** Debido a que no es posible automatizar completamente todo el sistema, siempre es necesario algún tipo de intervención humana. Todos los procesos y subsistemas que lo requieran se pueden poner en esta categoría.

   Éste módulo tiene una gran importancia en cuanto al diseño de interfaces, ya que obliga a crear una interfaz de usuario sencilla e intuitiva.

   b) ***Procesamiento:*** Se refiere a toda herramienta de procesamiento automatizado que no requieren de interfaces ni de conexiones con otras máquinas. Un ejemplo pueden ser herramientas de aprendizaje que permitan a la maquina descubrir nuevas rutas.

   c) ***Módulos de transformación de datos:*** Cuando la información de entrada proviene de fuentes "externas" a la aplicación, es necesario realizar una transformación del formato de éstos para que puedan ser guardados en la base de datos.

   Obtener el formato final puede ser una tarea simple cuando se está tratando con información proveniente de fuentes similares. Sin embargo, cuando la información proviene de fuentes que no respeten los formatos y estándares, puede generar una enorme carga sobre los servidores al tratar de convertir su formato y por ende disminuir la velocidad de respuesta del sistema.

   d) ***Procesamiento de salida:*** Toda la información que se encuentra guardada dentro de la base de datos debe de ser procesada antes de poder ser mostrada. El procesamiento más simple es leer de la base de datos los registros y enviarlos, mientras que un procesamiento avanzado puede ser la transformación a XML compatible con mapas como Google Maps.

4. **Módulos administrativos:**

   a) ***Sistema de Gestión de dispositivos:***[3] Este sistema se encargará de registrar y

[3]Dispositivo: Los dispositivos, dentro de éste trabajo de tesis, serán definidos como componentes electrónicos que leen o escriben datos en medios o soportes de almacenamiento a través de una red de datos.

controlar los dispositivos que pueden introducir información a la API así como el control de privilegios.

b) ***Sistema de gestión de vehículos:***[4] Permite relacionar a cada socio y chofer del sistema con sus respectivos vehículos, también genera la relación con sus placas, número de concesión, etc.

c) ***Sistema de gestión de puntos de control:*** Permite ingresar de manera manual puntos de control en la ruta en donde haya habido una falla de un vehículo o dispositivo. Esto ayuda a que se pueda mantener un registro completo de las posiciones no importando la situación y a facilitar la administración.

d) ***Reportes:*** Permite obtener reportes del servidor tales como intentos de acceso no autorizados, problemas con información de entrada, fallos críticos en el sistema, etc.

e) ***Administración de usuarios y roles:*** Permite registrar a los usuarios que van a utilizar el sistema y sus roles dentro del sistema. Esta funcionalidad es exclusiva del administrador del proyecto.

3.2.3. Arquitectura propuesta

En la figura 3.1 se muestra un esquema simplificado de cómo se ven los módulos que componen el proyecto, sin incluir los módulos administrativos, y sus conexiones entre ellos.

Como se puede notar, la API funciona como puerta de entrada al sistema ayudando a reducir posibles puntos inseguros, aislando los sistemas internos de los sistemas externos y dando un punto de conexión común a nuevos módulos que deban ser agregados.

3.3. Ambiente de trabajo

El prototipo de este prototipo que se propone en este trabajo de tesis se desarrolló con herramientas de software libre, siempre que sea posible, tanto para los entornos de desarrollo como para la bases de datos.

El ambiente en el que se realizaron las pruebas tiene las siguientes características:

1. **Servidor:**

   a) 3GB de memoria RAM

   b) 2 core CPUs

   c) Zona Horaria Ameria/Mexico_City

   d) Ubuntu 12.04 LTS (2013/07)

[4]Vehículo: Un vehículo es un medio de locomoción que permite el traslado de un lugar a otro. Cuando traslada a animales u objetos es llamado vehículo de transporte, como por ejemplo el tren, el automóvil, el camión, el carro, el barco, el avión, la bicicleta y la motocicleta, entre otros.

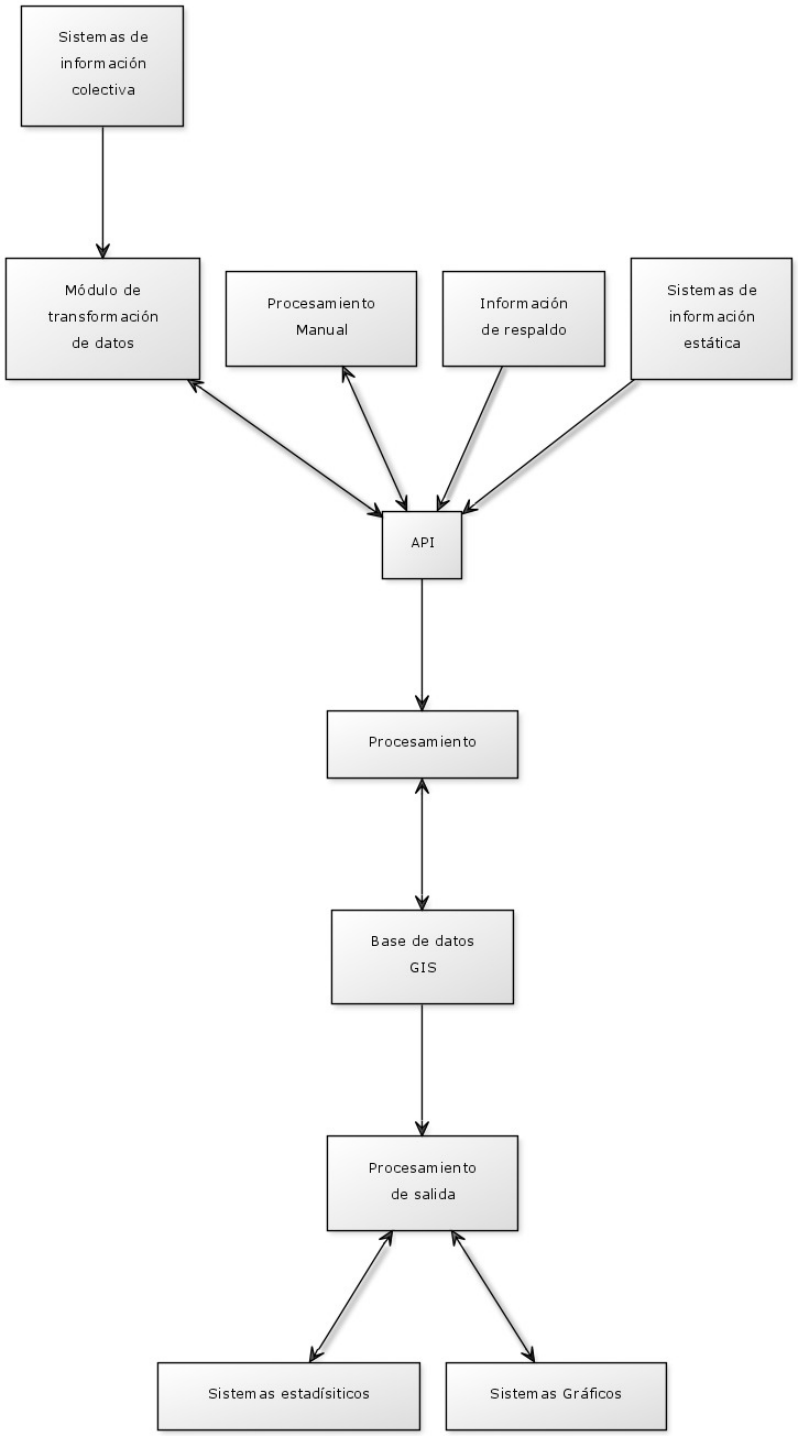

Figura 3.1: Interconexión entre los módulos del sistema sin módulos administrativos

1) Procesador AMD x64
2) Linux 3.5.0-41-generic

e) Modificaciones

1) Aumento el límite a 10000 en /etc/security/limits.conf
2) Actualización del archivo /etc/sysctl.conf para manejar un número mayor de conexiones TCP

2. **Redes:**
    a) Conexión a localhost a través de puerto 4000
    b) Interfaz de red interna 127.0.0.1
3. Interprete de Python 3.2
4. Motor de bases de datos PostgresSQL 9.1.9
5. Librería PostGIS 2.0.1 r9979
6. Librería GEOS 3.3.8-CAPI-1.7.8
7. Librería PROJ Rel. 4.7.1
8. Librería LIBXML 2.7.8
9. Sistema celular android 2.3 con GPS y conexión de datos Wi-Fi

3.3.1. Funcionalidad esperada de los módulos

En esta sección se da un resumen del comportamiento esperado de cada uno de los módulos propuestos dentro del proyecto.

1. **Sistema de gestión de dispositivos:**
    a) Permitir registrar los dispositivos que se están utilizando
    b) Permitir consultar los dispositivos y los datos asociados a ellos
    c) Permitir la edición de la información de los dispositivos
    d) Permitir la eliminación de la información de los dispositivos
2. **API:**
    a) Permitir agregar nuevos registros con la posición de los vehículos
    b) Permitir consultar registros de la posición de los vehículos
    c) Permitir consultar información de los vehículos
    d) Permitir acceder a la información de la base de datos en tiempo real
    e) Conectar diferentes módulos y funcionalidades
3. **Sistema de gestión del transporte:**
    a) Permitir registrar vehículos y asociarlos a un dispositivo
    b) Permitir consultar vehículos y la información asociado a ellos
    c) Permitir editar la información de un vehículo

d) Permitir eliminar la información de un vehículo

4. **Sistema de gestión de puntos de control:**

   a) Permitir registrar puntos de control

   b) Permitir consultar puntos de control

   c) Permitir editar puntos de control

   d) Permitir eliminar puntos de control

5. **Administración de usuarios y roles**

   a) Permitir incluir usuarios dentro de grupos específicos con permisos configurados

   b) Permitir registrar nuevos usuarios dentro de la plataforma

   c) Permitir editar usuarios dentro de la plataforma

   d) Permitir eliminar usuarios dentro de la plataforma

   e) Permitir consultar usuario dentro de la plataforma

   f) Permitir modificar los roles y grupos a los que pertenece un usuario

6. **Sistema gráfico:**

   a) Permitir visualizar el tráfico en tiempo real en un mapa

   b) Mostrar el resultado del sistema de búsqueda en un mapa

7. **Base de datos:**

   a) Registra las coordenadas de la posiciones de los vehículos

   b) Ejecutar disparadores programados que realizan procesamiento con la información al momento de realizar alguna acción sobre ella

   c) Permitir la consulta de información basado en roles

   d) Realizar cálculos espaciales sobre la información y retornarla a las aplicaciones que lo piden

8. **Reportes:**

   a) Generar reportes sobre errores en el sistema y los guarda en archivos de registro

   b) Generar alarmas cuando un error crítico aparece

   c) Permitir consultar errores por fecha, hora o tipo de reporte

3.4. Experiencia de usuario

La experiencia de usuario se puede definir como el nivel de satisfacción que un usuario obtiene al utilizar un sistema y es tan importante como la función principal del sistema. [52] Si la experiencia es mala, el sistema es difícil de manejar, tiene una curva de aprendizaje muy grande o es muy complejo y tardado de operar diariamente es posible que no sea práctico y termine sin ser utilizado.

A lo largo de este proyecto se diseña un sistema que presente las siguientes características:

1. **Interfaces de usuario intuitivas:** Las interfaces de usuario serán simples y limpias. No deben de suponer la necesidad de consultar la documentación cada vez que son utilizadas.

2. **Curva de aprendizaje suave:** No debe ser necesaria una gran capacitación para comenzar a utilizar el sistema. Así mismo, la complejidad de los conocimientos que se requiere variará dependiendo con qué módulos del proyecto el usuario interactúa.

3. **Facilidad de uso:** Cada módulo que forme parte del sistema debe ser práctico y sencillo de entender. De esta manera se puede revisar qué módulos y funcionalidades ya existen evitando repeticiones.

4. **Disponibilidad:** Los servidores y bases de datos, así como su arquitectura, deben ser pensados para un trabajo 24/7. Para realizar esto se puede trasladar el sistema a la nube o utilizar una arquitectura de cluster.

5. **Tiempo de respuesta:** El tiempo de respuesta del sistema puede llegar a ser variado dependiendo de qué tan ruidoso es el canal de comunicación, cuántos dispositivos conectados a la misma antena y transmitiendo al mismo tiempo hay, la tasa de transmisión de la red, la cantidad de datos que se pida a la bases de datos, etc. Es por ello que se debe intentar realizar operaciones asíncronas que aunque complican la programación del sistema, permiten operaciones no bloqueantes.

6. **Carga de trabajo:** La arquitectura debe estar pensada en un escalamiento horizontal en un futuro. Una vez más se podría pensar en la computación en nube para dar solución a esto.

7. **Degradación:** El error mínimo aceptable que pueden llegar a tener los datos puede variar dependiendo de condiciones externas al sistema. Esto implica que si existe éste error será el dispositivo de seguimiento el que tendrá que realizar las correcciones de la información antes de volver a enviarla.

8. **Consumo de recursos:** Actualmente es posible conseguir fácilmente una computadora con características suficientes para este tipo de sistemas de recepción. Sin embargo, es

importante mencionar que es recomendado como mínimo 200 Gb de almacenamiento y 1 Gb de Memoria RAM con un procesador a 1.8 Ghz y una tarjeta de red de 100 Mbps[5].

3.5. Análisis del sistema

Los flujos de información que tendrá la aplicación final se analizaron con los requerimientos de negocio usando esquemas de alto nivel. Los módulos que se analizaron en esta sección son aquellos considerados *críticos* o *necesarios* y se describen a continuación.

3.5.1. Flujos de módulos administrativos

Los módulos se describen para sólo administradores aunque los flujos deberían ser similares para los usuarios, exceptuando, tal vez, en la interfaz que utilizan o en las actividades que pueden realizar. En la figura 3.2 se muestran un mapa de las diferencia en las actividades que podría haber entre ellos.

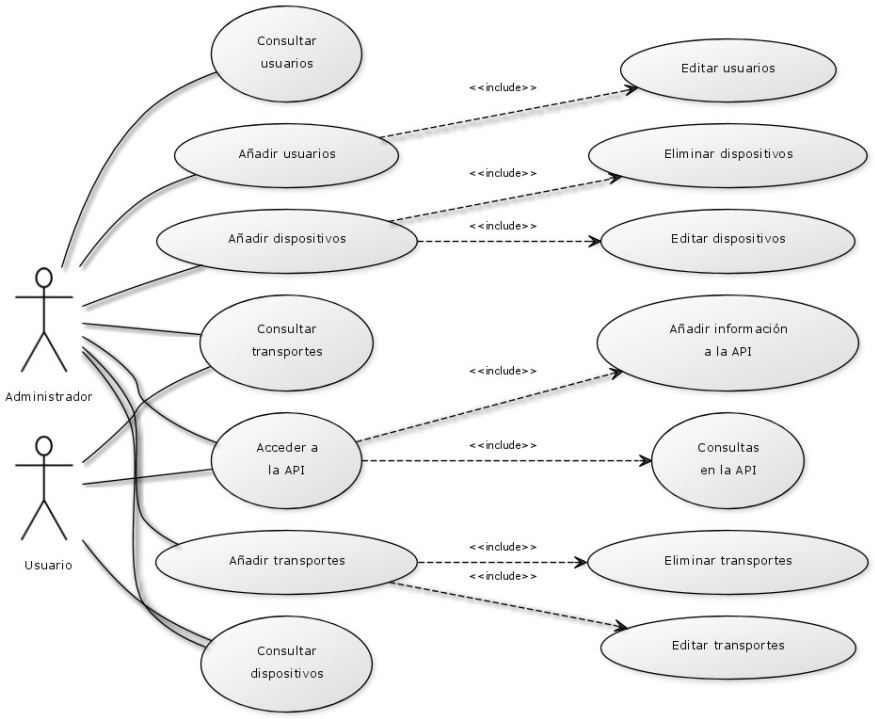

Figura 3.2: Actividades permitidas para los usuarios y administradores

Cada uno de los subflujos mencionan los pasos a seguir de manera que los módulos puedan ser utilizados con interfaces gráficas o a través de la API. En los diagramas de uso que se muestran en estos subflujos, como el de la figura 3.3, se muestra el cómo éstos pasos trabajarían utilizando una interfaz gráfica.

[5]Datos obtenidos a partir de un promedio en las especificaciones de la base de datos y el servidor utilizados

3.5.1.1. Sistema de gestión de dispositivos

Para que se pueda registrar un nuevo dispositivo se debe tener permisos de administrador y relacionarlo con el vehículo correspondiente a través de un identificador.

1. **Consultar dispositivos:**

   a) Se realiza una petición a la URL que contiene la colección[6] "Dispositivos"

   b) El sistema envía el listado con todos los dispositivos registrados

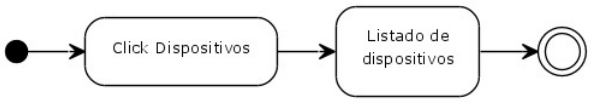

Figura 3.3: Representación del flujo de consulta de dispositivos para interfaces gráficas

2. **Registrar dispositivos:**

   a) Se realiza una petición a la URL que contiene la colección "Dispositivos"

   b) El administrador envía una petición HTTP[7] con la información del dispositivo

   c) El sistema genera un identificador para el dispositivo

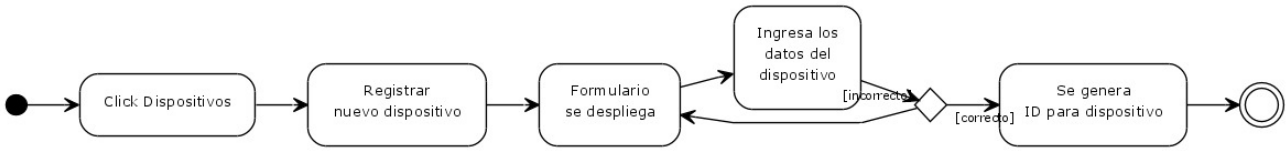

Figura 3.4: Representación del flujo de registro de dispositivos para interfaces gráficas

3. **Editar dispositivos:**

   a) Se realiza una petición a la URL que contiene el documento del "Dispositivo" que se quiere editar.

   b) El administrador envía una petición HTTP con la nueva información del dispositivo

   c) Si la información es correcta el dispositivo se actualiza, si es incorrecta se envía un mensaje de error

4. **Eliminar un dispositivo:**

   a) Se realiza una petición a la URL que contiene el documento del "Dispositivo" que se quiere eliminar

   b) El administrador envía la petición HTTP al servidor

   c) Si el administrador tiene los permisos suficientes, el dispositivo es eliminado.

[6] Términos como colección, recurso o documento son explicados en detalle en el capítulo 5 y dentro del glosario

[7] La petición que se realice para cada acción será descrita en la capítulo 5

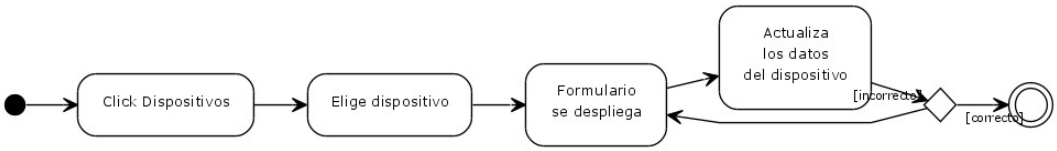

Figura 3.5: Representación del flujo de actualización de dispositivos para interfaces gráficas

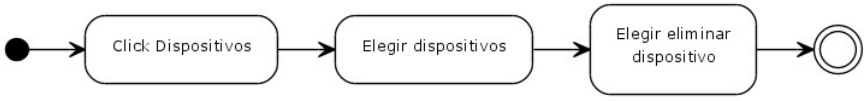

Figura 3.6: Representación del flujo de eliminación de dispositivos para interfaces gráficas

3.5.1.2. Sistema de gestión del transporte

Para que se pueda registrar un nuevo vehículo es necesario tener permisos de administrador y el vehículo no debe de existir en la base de datos.

1. **Consultar vehículos:**

   a) Se realiza una petición a la URL que contiene la colección “Vehículos”

   b) El sistema envía un listado con todos los vehículos registrados

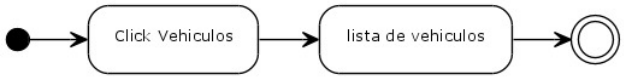

Figura 3.7: Representación del flujo de consulta de vehículos para interfaces gráficas

2. **Registrar vehículos:**

   a) Se realiza una petición a la URL que contiene la colección “Vehículos"

   b) El administrador envía una petición HTTP con la información del vehículo

   c) El sistema verifica que la placa y el número de vehículo sean únicos

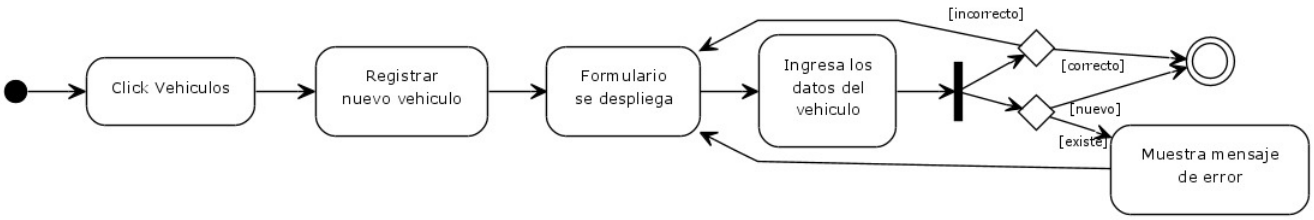

Figura 3.8: Representación del flujo de registro de vehículos para interfaces gráficas

3. **Editar vehículos:**

   a) Se realiza una petición a la URL que contiene el documento “Vehículo” a editar

b) El administrador envía una petición HTTP con la nueva información del vehículo

c) Si la información es correcta el vehículo se actualiza, si es incorrecta se envía un mensaje de error.

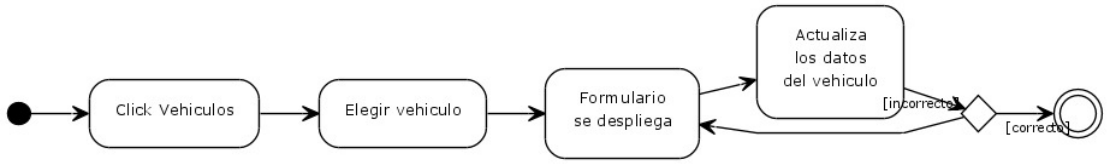

Figura 3.9: Representación del flujo de actualización de vehículos para interfaces gráficas

4. **Eliminar un vehículo:**

a) Se realiza una petición a la URL que contiene el documento "Vehículo.ª eliminar

b) El administrador envía una petición HTTP al servidor

c) Si el administrador tiene los permisos suficientes, el dispositivo es eliminado

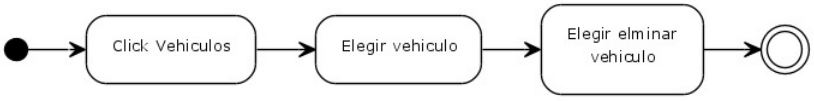

Figura 3.10: Representación del flujo de eliminación de vehículos para interfaces gráficas

Entre los flujos que no estarán permitidos en el prototipo son:

a) **Registrar placas iguales:** Las placas se utilizan para identificar cada uno de los vehículos como parte de la llave primaria y por ende deben de ser únicas

3.5.1.3. Administración de usuarios y roles

Para que un usuario pueda ser registrado, debe de existir un usuario administrativo que le de ese permiso. Una vez que el usuario ha sido insertado, actualizado o eliminado, debe ser posible ver la nueva lista de usuarios.

1. **Consultar usuarios:**

a) Se realiza una petición a la URL que contiene la colección Üsuarios"

b) La lista de usuarios se muestra

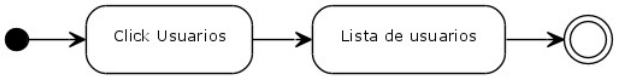

Figura 3.11: Representación del flujo de consulta de usuarios para interfaces gráficas

2. **Registrar Usuario:**

   a) Se realiza una petición a la URL que contiene la colección ”Usuarios“

   b) El administrador envía una petición HTTP con la información del usuario

   c) El sistema revisa que el usuario no exista y genera un identificador

.

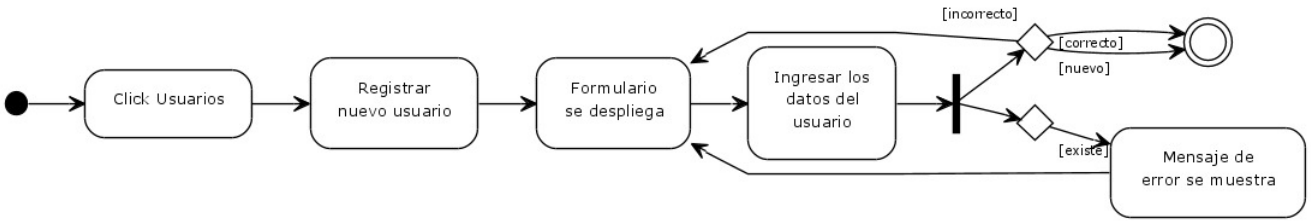

Figura 3.12: Representación del flujo de registro de usuarios para interfaces gráficas

3. **Editar usuario:**

   a) Se realiza una petición a la URL que contiene el documento ”Usuario“ a editar

   b) El administrador envía una petición HTTP con la nueva información del usuario

   c) Si el usuario existe y la información es correcta la base de datos se actualiza actualizada, si es incorrecta se envía un mensaje de error.

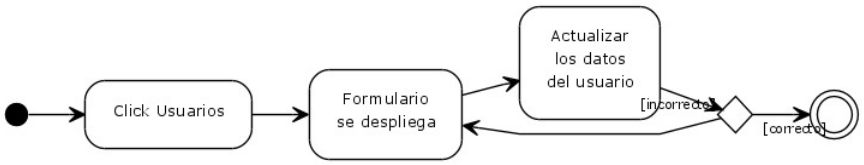

Figura 3.13: Representación del flujo de actualización de usuarios para interfaces gráficas

Entre los flujos que no estarán permitidos en el prototipo son:

1. **Registrar usuarios iguales:** El sistema no permitirá registrar usuarios iguales dentro de la base de datos. Si el usuario existe un mensaje es enviado.

2. **Eliminar usuarios:** Ningún administrador o usuario podrán eliminar una cuenta. Las cuentas se desactivarán y no permitirán el acceso al sistema.

3. **Edición de usuarios desactivados:** Si un usuario se encuentra desactivado, no podrá modificarse sus datos hasta que no se reactive de nuevo.

3.6. Requerimientos de la solución

Cuando se analizó la arquitectura fue necesario diferenciar los servicios[8] que son públicos de los que son privados. También se consideró que algunos servicios pueden ser utilizados por una agencia[9] en particular, mientras otros pueden ser utilizados por todas las agencias.

La razón para esta distinción es debido a que generalmente se tienen diferentes requerimientos en términos de seguridad y estabilidad a lo largo de los servicios. Mientras que los servicios privados o locales pueden no requerir de procesos de seguridad muy avanzados, los servicios públicos usualmente deben tener condiciones muy especiales de seguridad. Al mismo tiempo mientras que los servicios públicos, los cuales puede dar información también a personas anónimas y por ello servir mayor cantidad de información, deben tener altos requerimientos de estabilidad y robustez los servicios internos pueden trabajar sobre interfaces más inestables que permitan modificarlas y administrarlas de una manera más sencilla. Finalmente los servicios públicos también deben de tener acuerdos de servicios distintos, un ejemplo es la disponibilidad del sistema que debe de ser 24 horas al día 7 días a la semana.

Otras diferencias que pueden existir entre servicios privados y servicios públicos son las tecnologías de interfaces, en los primeros pueden permitirse protocolos propietarios para aumentar la velocidad pero no en los segundos que deben sólo trabajar con interfaces que sean estándares.

Es por ello que para la arquitectura propuesta en este trabajo se pensó en sólo una interfaz pública que permita dividir los servicios privados de los públicos. En esta arquitectura, la API funciona como un buffer (ESB) que permite a todos los servicios internos trabajar independientemente de los servicios externos. Esto da como resultado la posibilidad de modificar ambos lados de la API sin generar problemas de interconexiones, también disminuye dramáticamente el número de conexiones punto a punto que se debe de manejar ya que ahora todas las agencias se conectan a un sólo punto y todos los controladores, incluidos algunos que puedan estar especializados en alguna agencia, se comunican con estas a través de la API.

3.6.1. Servicios de escritura y de lectura

Dentro de cada uno de los servicios, ya sean públicos o privados, es necesario también distinguir entre cuáles son de escritura(servicios que modifican información) y cuáles son de lectura (servicios que no modifican información).

Por definición los servicios de lectura deben de tener atributos especiales y generalmente deben de introducir pocos problemas, esto se puede lograr creando servicios idempotentes[10] y

[8]Servicio: Un servicio es un módulo o un conjunto de módulos que presentan una funcionalidad completa o son una aplicación finalizada

[9]Agencia: Institución de la administración pública que se encarga de ejecutar y administrar algún servicio público

[10]Idempotencia: Es la habilidad de los servicios para manejar mensajes iguales que llegan más de una vez, de

haciendo que no se requiera transacciones atómicas[11].Los servicios de escritura, por otro lado, siempre requieren modificaciones de la información que sean seguras y que estén documentadas. Por esta razón, los servicios de escritura requieren información técnica adicional que identifique qué mandó a llamar el servicio.

Los servicios de lectura generalmente son menos estables que los servicios de escritura, la razón es que los primeros no afectan procesos esenciales mientras que los segundos pueden modificar atributos que sean necesarios para otros servicios.

También existe una razón administrativa para dividir los procesos de escritura y lectura. Cuando en un sistema se realizan actualizaciones o reparaciones una parte de la infraestructura queda inactiva por algunas horas. Sin embargo, es posible copiar la información a sistemas de respaldo que sólo permitan acceso de lectura mientras se continua con los servicios, de esta manera sólo los procesos de escrituras son apagados lo que permite seguir proveyendo parte del servicio.

3.7. Políticas de diseño y documentación

En esta sección se presentan las políticas que se consideraron necesarias cuando se diseñó el servicio web. Estas políticas están divididas en tres categorías:

1. Documentación
2. Seguridad
3. Testeo y Comportamiento

Que son tres secciones necesarias que se deben de tomar en cuenta desde el inicio del diseño y desarrollo del sistema. Estas políticas se explicarán a través de cuatro propiedades:

1. **Nombre:** El nombre con el que se identifica la política. Debe de ser lo menos ambiguo posible.
2. **Descripción:** Como su nombre lo indica, explica el funcionamiento de la política.
3. **Beneficios:** Justifica la aplicación de la política en el diseño del sistema.
4. **Implicaciones:** Describe cómo afecta la política a aquellos que deben cumplirla.

Para un manejo más sencillo de las políticas, se utilizarán las siguientes abreviaturas como nombres:

1. POL_DOC_# Políticas relacionadas con documentación y diseño

manera que no se realice la misma operación múltiples veces

[11]Atomicidad: Es la habilidad de un servicio o programa de regresar a un estado previo si la transacción no se logró correctamente.

2. POL_SEG_# Políticas relacionadas con seguridad

3. POL_TEST_# Políticas relacionadas con testeo y comportamiento

3.7.1. Propuestas de políticas para la documentación y diseño

1. **Los servicios deben de auto-documentarse**

Política: POL_DOC_1	**Crear documentos auto-documentados**
Descripción	Todos los servicios a través de la API así como las URL que se utilicen, debe de ser auto-documentados. Esto quiere decir que no debe haber necesidad de complejos manuales para poder utilizar el servicio.
Beneficios	Esto permite a los usuarios el poder utilizar el servicio sin la necesidad de una documentación, ya sea a través de libro o en linea, demasiado extensa y compleja. Mantener la documentación pequeña ayuda a reducir el tiempo y trabajo que se necesita para desarrollar y actualizar el sistema.
Implicaciones	Los servicios deberían de utilizar relaciones, nombres de métodos y URL que permitan entender cómo los servicios deberían de ser utilizados. Un documento de texto debe de ser guardado junto con el servicio para definir el servicio donde explica cómo se debe de utilizar su API.

Tabla 3.2: Política POL_DOC_1

2. **Se deben de utilizar las representaciones de datos más aceptados:** Debido a que es probable que este sistema tenga que interactuar con otros es importante mantener una funcionalidad que sea aceptada ampliamente dentro de la industria.

Política: POL_DOC_2	**Representación de datos comunes**
Descripción	Si existen estándares ya sean nacionales o internacionales que sean ampliamente utilizados, es importante que el sistema cumpla con ellos para poder trabajar con estas entidades. También se debe de asegurar que estos son definidos dentro de cada uno de los submódulos que interactúan con el mundo exterior.
Beneficios	Cuando se siguen los estándares la integración con otros servicios se vuelve mucho más sencilla y por ello menos costosa. Es mucho más sencillo modificar la representación de los datos cuando se encuentra la necesidad. Es por ello que el capítulo 5 se analizan los actuales tipos de representación más usados y se realiza un análisis comparativo.

Continua siguiente página

Tabla 3.3 – continuación

Política: POL_DOC_2	**Representación de datos comunes**
Implicaciones	Antes de definir nuevos módulos o servicios para aumentar las funcionalidades del sistema, es importante hacer un estudio de los formatos para representación de datos que existen y que permiten a los sistemas trabajar en ambientes distribuidos (XML es un ejemplo de esto), así como analizar las implicaciones de utilizar cada uno ya que pueden llegar a afectar el comportamiento del sistema. En el caso de que no existiesen aún estándares o de que no exista uno que sea ampliamente utilizado dentro de la industria, entonces es importante definir cuál será utilizado a través de la documentación.

Tabla 3.3: Política POL_DOC_2

3. **Debe de existir un servidor o un módulo independiente que se dedique a manejar cuentas de usuario:** Cuando los sistemas crecen y comienzan a aumentar los servicios, así como a unir nuevos sistemas externos, es muy común terminar con múltiples bases de datos manejando cuentas de usuario, generando información redundante y haciendo lento al sistema al tener que buscar a través de todas ellas.

Política: POL_DOC_3	**Servidor centralizado de usuarios**
Descripción	Todos los servicios deben de tener un sistema de manejo de cuentas único. Antes de que un nuevo servicio sea desarrollado o incluido, debe de transmitirse toda la información entre los módulos que controlan las cuentas de usuario hacia el que fue considerado como elemento final. También debe de asegurarse de que el servicio escogido tenga todo lo necesario para que el nuevo sistema pueda ser migrado. En caso de que esto no sea así debido a la falta de funcionalidades, siempre es mejor añadirlas al sistema final a tener redundancia de servicios. Sin embargo, esto último puede implicar que el servicio esté mal diseñado.
Continua siguiente página	

Tabla 3.4 – continuación

Política: POL_DOC_3	**Servidor centralizado de usuarios**
Beneficios	Es posible evitar la necesidad de crear nuevos servicios o conectores que aumentan innecesariamente la complejidad del sistema y por ende el costos y tiempo de mantenimiento. Introduciendo sistemas especializados en ciertas taras hace que todos los servicios se puedan conectar fácilmente al sistema y los desarrolladores no necesiten reconstruir algunas funcionalidades que necesitan desde cero. Con esto se logra crear equipos de desarrollo independientes entre sí y también se logran arquitecturas más modulares en las que los subsistemas son independientes unos de otros.
Implicaciones	Antes de que cualquier servicio sea creado o anexado los desarrolladores deben de revisar si todos los sistemas necesarios para su trabajo existen y si no existe ya un servicio que resuelva su problema. El servicio central debe de estar disponible mientras se realiza el diseño del sistema.

Tabla 3.4: Política POL_DOC_3

4. **Cada cambio dentro de la API que modifique su comportamiento debe de ser puesto en una versión nueva y se debe dar soporte al menos a la versión anterior de esta:** Cuando se realizan modificaciones a la API se pueden generar problemas en las aplicaciones que dependen del sistema al ya no cumplir por completo con las especificaciones de la nueva API. Es por ello que es importante siempre tener versiones de ésta correctamente divididas y dar soporte a al menos una versión anterior.

Política: POL_DOC_4	**Soporte de múltiples versiones para la API**
Descripción	Debe de ser posible soportar dos versiones a la vez dando la posibilidad a los usuarios de actualizar su sistema. Cuando una nueva versión es finalizada se debe de revisar si el cambio crear una incompatibilidad con las aplicaciones actuales generando la necesidad de un nuevo número de versión.
Beneficios	La razón de ésta política es que no se puede esperar que los usuarios actualicen inmediatamente sus aplicaciones en cuanto la API sea actualizada, principalmente si se permiten aplicaciones de terceros. También se debe de contemplar la posibilidad de que los servicios internos que dependen de la API requieran actualizarse al mismo tiempo. Para permitir a los desarrolladores y a los usuarios una actualización suave de sus sistemas, se debe soportar la versión anterior.

Continua siguiente página

Tabla 3.5 – continuación

Política: POL_DOC_4	**Soporte de múltiples versiones para la API**
Implicaciones	Debe de existir una descripción de las versiones que explique detalladamente los cambios realizados junto con la descripción del ciclo de vida o el tiempo que se le dará mantenimiento a cada una de las versiones.

Tabla 3.5: Política POL_DOC_4

3.7.2. Propuestas de políticas de seguridad

1. **El canal de comunicación debe ser seguro:** La información considerada importante que es trasladada desde los puntos terminales hasta el servidor debe de ser cifrada o enviada a través de un canal seguro utilizando protocolos de seguridad como HTTPS o FTPS.

   Si la información trasladada no es importante estas medidas se pueden ver como opcionales dentro del propio sistema, ya que aumenta mucho la cantidad de ciclos de proceso necesarios para trabajar con la misma cantidad de información.

Política: POL_SEG_1	**Cifrar los canales para información crítica**
Descripción	Toda la información que sea importante o sensible debe de ser cifrada antes de ser enviada a través de las redes. El cifrado debe existir en el sitio web y en la API.
Beneficios	Los servicios web utilizados por usuarios y servicios de terceros requieren que exista confianza de que los datos se encuentran seguros y que solamente el proveedor del servicio tendrá acceso a ellos. Si esto no se cumple se puede llegar a tener un problemas desde una fuga de usuarios hasta sanciones legales y económicas. Proveyendo de servicios cifrados se asegura al usuario que sus datos estarán seguros.
Implicaciones	Cuando un servicio se desarrolla se debe estudiar si manejará información sensible, en caso de que éste sea el caso se debe diseñar basado en protocolos seguros desde el inicio y contemplar la necesidad de actualizar los certificados de seguridad. Si se van a manejar certificados a nivel cliente es necesario tomar en cuenta la administración y distribución de esos certificados también.

Tabla 3.6: Política POL_SEG_1

2. **Validar la integridad de los mensajes y la procedencia de estos:** Actualmente existen una gran cantidad de fuentes electrónicas que pueden fácilmente proveer información a través de GPS como receptores GPS, celulares e incluso algunos tipos de computadoras. La falsificación de la información que se envía al receptor puede crear una sobrecarga en las bases de datos dejando el sistema inutilizado.

Política: POL_SEG_2	**Validar la integridad de los mensaje y su procedencia**
Descripción	Asegurar que el origen de todos los mensajes que llegan al servidor sea de fuentes confiables y que la información sea real en todo momento.
Beneficios	Controlar las fuentes de información del sistema, así como analizar los paquetes, permite evitar ataques propio como buffer overflow y la saturación de las base de datos. También evita realizar cálculos innecesarios y erróneos debido a que la información no cumpla con el formato o que sea falsa.
Implicaciones	Entender y diseñar un método para crear una firma única para cada uno de los proveedores, la cual es enviada en cada petición junto a la información permitiendo identificarlos. Proveyendo las librerías o herramientas necesarias a los clientes para que firmen sus paquetes permite asegurar la calidad de las firmas, tener un mayor control de la información que se mueve en el sistema y ayuda a controlar versiones de la API fácilmente. Finalmente los sistemas deben de tener un módulo de entrada que valide la información antes de ser procesada.

Tabla 3.7: Política POL_SEG_2

3. **Manejo de cuentas unificadas para todos los servicios:** Esta es una tendencia que se está viendo en todo el mundo dentro de la industria. Empresas como Google o Facebook utilizan sistemas de autenticación únicos para todos sus sistemas. Esto beneficia el sistema al no tener un sistema de autenticación independiente para cada uno de los servicios reduciendo la cantidad de tiempo que se utiliza para actualizados y administrados. Al mismo tiempo evita ataques XSS[12] a través de códigos unificados que actualmente son sencillos de generar en lenguajes como Javascript, Ruby, Java o Python.
   Esto no debe de ser confundido con el sistema con la política POL_DOC_3 [Tabla 3.4].La política propuesta aquí hace referencia a un recurso en la web que unifique la autenticación como una URL para la API o una página única para identificarse para el usuario y que permita el manejo de sesiones entre servicios.

[12]Cross Site Scripting: Es un ataque en la que un atacante inyecta un script del lado del cliente permitiéndole traspasar controles de acceso

Política: POL_SEG_3	**Manejo de cuentas unificadas para todos los servicios**
Descripción	Todas las aplicaciones y servicios deben de tener un mecanismo de autenticación centralizado.
Beneficios	Permite realizar fácilmente un seguimiento del comportamiento de los usuarios con la plataforma y la API mientras utilizan varios servicios al mismo tiempo. También disminuye los costos de desarrollo al no ser necesario que cada sistema maneje sus propios esquemas.
Implicaciones	Los sistemas de autenticación mencionados en ésta política y en la política POL_DOC_3 deben ser implementados antes que cualquier otro módulo o servicio y deben ser accesibles siempre que sean necesarios.

Tabla 3.8: Política POL_SEG_3

Política: POL_SEG_4	**Sistemas centralizados de manejo de permisos**
Descripción	El sistema central de autenticación debe de determinar los roles y permisos de los usuarios.
Beneficios	Los sistemas de autenticación permiten especificar los roles y permisos que un usuario tiene dentro de cada servicio, permitiendo tener tantos roles como servicios existan. Teniendo un sistema centralizado se reduce el número de sitios en los que se deben administrar estos roles, en lugar de tener un sistema de administración por servicio, con un sistema centralizados se pueden manejar todos desde la misma interfaz.
Implicaciones	Cuando se diseña nuevos servicios se debe considerar los permisos y autorizaciones que tendrán los usuarios. Todos los servicios que requieran autorización no necesariamente deben seguir las misma reglas, esto implica que un usuario administrador en un servicio puede no ser administrador en otros servicios.

Tabla 3.9: Política POL_SEG_4

3.7.3. Propuestas de políticas de testeo y comportamiento

1. **La información del GPS debe ser procesada en un tiempo no mayor a 100ms:** Debido a la necesidad de que el sistema funcione en tiempo real el tiempo de procesamiento no debe superar una marca que sea percibida como tal. En la industria, el límite de 100 ms es muy utilizado para poder dar a los usuarios la sensación de que la petición y respuesta son en tiempo real.

Política: POL_TEST_1	**Procesar mensajes en menos de 100 ms**
Descripción	El tiempo promedio de procesamiento de los paquetes en el servidor debe ser menor a 100ms
Beneficios	Cuando un sistema tarda mucho tiempo en procesar una petición la cola de paquetes en el buffer se vuelve cada vez más grande y difícil de controlar. Si se requiere escalar el sistema, la cantidad de tiempo que los paquetes tardan en pasar de la red a la base de datos debe ser disminuida. A esto se le suma la necesidad de los usuarios de conocer esta información casi en el instante que es generada para que sea útil. Si la información llega demasiado tarde deja de tener valor alguno y el sistema simplemente no sirve.
Implicaciones	Tener un sistema de monitoreo que revise el tiempo promedio de procesamiento entre el momento que se recibe la petición y el momento en el que se envía la respuesta.

Tabla 3.10: Política POL_TEST_1

2. **Medir el comportamiento de de los servicios en tiempo real:** Visualizar los problemas del sistema en tiempo real puede prever problemas antes de que dañen el sistema por completo. Esto permite reducir el costo de mantenimiento y reparaciones.

Política: POL_TEST_2	**Medir los servicios en tiempo real**
Descripción	Todos los servicios deben de estar conectados al sistema de monitoreo y deben de enviar su información siempre que les sea posible.
Beneficios	Cuando se es capaz de revisar todos los servicios se puede reaccionar a los problemas en el momento que estos ocurren. Permite analizar si el sistema reacciona dentro de su comportamiento esperado en todo momento.
Implicaciones	Todos los servicios deben de tener un módulo que envíe los eventos ocurridos en el servicio en todo momento. Los eventos necesitan ser procesados por un sistema de monitoreo central como un BAM[13] o un módulo de monitoreo que reaccione a ciertas señales y pueda avisar a los administradores en caso de fallo.

Tabla 3.11: Política POL_TEST_2

3. **Los servicios deben poderse escalar horizontalmente:** Para que un sistema sea

[13]Bussines Activity Monitor (BAM): Permite revisar todos lose servicios en un sistema distribuido heterogéneo a través de una interfaz XML

escalable no basta con que la arquitectura en general lo sea, también deben de ser sus servicios y módulos. Para un sistema como este que se espera crezca es importante diseñar a cada uno de ellos para escalar horizontalmente.

Política: POL_TEST_3	**Los servicios deben poderse escalar horizontalmente**
Descripción	Todos los servicios deben poder escalar horizontalmente. De ser posible se deben de poder ejecutar en la nube.
Beneficios	La utilización de los servicios se espera que crezca mucho si se comienzan a incluir otras ciudades y pueden necesitar una inversión grande en infraestructura en ése momento. Para mantener los gastos lo más bajo posible se pueden diseñar los servicios para ser escalables desde un inicio, lo que ayuda a no tener que rediseñar el sistema cada vez que se requiera escalar.
Implicaciones	Los servicios requieren una arquitectura muy bien estudiada antes de poder ser construidos y anexarlos al sistema.

Tabla 3.12: Política POL_TEST_3

4. **Todo el código debe estar debidamente respaldado con pruebas de testeo:**

Política: POL_TEST_4	**Todo el código debe estar debidamente respaldado con pruebas de testeo**
Descripción	Todos los servicios deben de contener sus respectivos segmentos de código de testeo que prueben todas las funcionalidades. Además deben de poder pasar tests generales y de integración continua cuando estos sean conectados al proyecto.
Beneficios	Los sistemas de testeo permiten encontrar problemas en el código a temprana edad e inclusive, utilizando técnicas como TDD[14] es posible reducir la cantidad de código que se utiliza en el sistema y encontrar errores en el momento de la programación.
Implicaciones	Cada capa, servicios y módulo debe ser testeado. Las herramientas y frameworks que se utilicen dependen del programador pero se debe asegurar que aquel que se escoja pueda cubrir lo mejor posible todos los escenarios. Antes de diseñar cualquier servicio o módulo se debe saber cuales son las métricas que se van a tomar en cuenta para medir en el comportamiento.

Tabla 3.13: Política POL_TEST_4

[14]Test driven development: Es una técnica de programación en la que el código de testeo se programa primero y después se crear el código del sistema que permite pasar el test.

Una vez que el esquema básico del prototipo se ha estudiado, es importante diseñar la estructura interna de las bases de datos. Esto es un aspecto muy importante del sistema, ya que son estas bases de datos las guardarán las información permitiendo utilizarla en otros proyectos a futuro. El siguiente capítulo se dedica exclusivamente a ésta tarea.

Capítulo 4

Bases de datos

Los sistemas de bases de datos son una de las partes más importantes en los sistemas de localización automática de vehículos ya que permiten mantener la información para ser utilizada en otras aplicaciones. En este capítulo se diseña la base de datos que se utiliza en el prototipo.

4.1. Introducción

Como se mencionó en capítulos anteriores, una base de datos se puede definir de una manera básica como un sistema que mantiene seguro una colección de datos. Para las nociones de este trabajo de tesis una base de datos se entenderá como como un contenedor electrónico ubicado dentro de un servidor que permite guardar información en forma ordenada y clasificarla dentro de un sistema de almacenamiento electrónico de archivos. A simple vista una base de datos geográfica no es diferente de una base de datos normal, ambas tienen una velocidad de respuesta muy similar y es posible modelar lo mismo en ambas. Para entender un poco mejor el porqué las bases de datos geográficas son necesarias entonces, se debe inspeccionar el mundo de las bases de datos especializadas en Big Data[1].

Las bases de datos NoSQL[2] son una realidad tecnológica viable actualmente. Estas bases de datos pueden, en teoría, guardar el mismo tipo de información que una base de datos relacional y al mismo tiempo ofrecer procesos especializados sobre la información más eficientemente de lo que es posible hacer en una base relacional directamente. El nacimiento de estas bases se debió a los problemas crecientes que se comenzaban a encontrar en los servicios web con el aumento de la cantidad de información que llegaba a un servicio. Basadas en la filosofía de escalamiento horizontal y sistemas distribuidos, estas bases tratan de resolver el problema del Big Data.

La necesidad de una base de datos geográfica se puede comparar a la necesidad de una

[1]Big Data: Es un término utilizado para describir un conjunto de datos tan grande y complejo que es difícil procesarlo con técnicas convencionales

[2]Base de datos NoSQL: Son bases de datos que permiten guardar información y leerla utilizando modelos menos consistentes que los que usa una base de datos relacional

base de datos NoSQL, la información geográfica no es más que un campo binario (BLOB[3]) que el software sabe interpretar. En realidad es posible hacer que una base de datos común tenga estos campos y programar rutinas que permitan realizar todas las operaciones matemáticas que se encuentran dentro de una base de datos geográfica. Sin embargo, al igual que en el caso de las bases NoSQL con respecto a las bases de datos relacionales, las bases de datos geográficas tienen habilidades especiales como entender la información que se le agrega y realizar operaciones espaciales sobre ella de manera más eficiente.

Las bases de datos geográficas también tienen extensiones para el lenguaje SQL que permiten realizar otro tipo operaciones geográficas sobre la información (como es el cálculo del área de una región), nuevos tipos de índices que ayudan a acelerar las búsquedas, varios tipos de tablas especializadas que manejan diferentes tipos de metadatos que se pueden encontrar en los sistemas GIS y los métodos que manejan siguen los estándares OGC.

4.1.0.1. Estándares OGC

Estos estándares son una series de recomendaciones dadas por el Open Geospatial Consortium (OGC). Definen una API cómo, un conjunto mínimo de extensiones GIS para el lenguaje SQL y otra serie de medidas que las bases de datos deben de cumplir para ser consideras compatibles con OGC.

Debido a la diversidad de la información geográfica y la variedad de fuentes que ésta puede tener, estos estándares son muy rigurosos. Esto asegura que cualquier sistema GIS en el planeta puede comunicarse con otros sistema GIS.

Existen algunas bases de datos que son espaciales pero no compatibles con OGC como MS SQl y MySQL. Todas las bases de datos compatibles con OGC deben de soportar dos tablas esenciales llamadas *geometry_columns* (Tabla 4.1)y *spatial_ref_sys* (Tabla 4.2), es por ello que la mayoría de los sistemas GIS utilizan la existencia de estas dos tablas para determinar si están hablando a un sistema GIS genuino.

La tabla *geometry_columns* es utilizada para guardar columnas dentro de la base de datos contienen información geográfica junto con el tipo de dato que es, el sistema de coordenadas, las dimensiones, ente otras cosas.

La tabla *spatial_ref_sys* es una lista de sistemas de referencia espaciales, o sistemas de coordenadas que son bien conocidas. Estos sistemas de coordenadas son los que define la localización en cualquier base de datos geográfica. Las entradas en esta tabla están indexadas utilizando un número conocido como EPSG[4] ID[5].

[3]Binary Large Object (BLOB): Es un conjunto de datos en forma binaria que se encuentran guardados dentro de una base de datos. Consisten principalmente de imágenes, videos, audio y otras formas de multimedia

[4]European Petroleum Survey Group (EPSG): Es una organización conformada por empresas que comercian con petroleo y gas en la Unión Europea y el Norte de África.

[5]European Petroleum Survey Group ID (EPSG ID): Las empresas que conforman este consorcio necesitaban resolver el principal problema cuando se buscan reservas de petroleo: posición a escala global. Las empresas utilizaban cada una sus escalas y sistemas de coordenadas propios.

Si la tabla *spatial_ref_sys* no existe o se encuentra vacía, no será posible mostrar la información en un mapa de una manera precisa o crear transacciones de información entre sistemas GIS.

Cada una de estas tablas tiene una estructura mínima que se establece en el estándar [31]. En los siguientes tablas se explica brevemente cuál es la función de cada uno de estos atributos.

1. **Geometry_columns**: La estructura de la tabla *geometry_columns* se muestra en el tabla 4.1

Atributo	**Descripción**
f_table_catalog	El nombre de la base de datos en la que está definida la tabla
f_table_schema	El esquema de la base de datos en la que está definida la tabla
f_table_name	El nombre de la tabla que guarda la información
f_geometry_column	El nombre de la columna que guarda la información
coord_dimension	Las dimensiones del sistema de coordenadas
srid	La referencia al ID del sistema de coordenadas en uso
type	El tipo de geometría guardado en la tabla

Tabla 4.1: Atributos de la tabla geometry_columns

Los atributos f_table_catalog, f_table_schema y f_stable_name son utilizados de diferentes maneras por diferentes bases de datos. La base de datos Oracle Spatial, por ejemplo, tiene una sola tabla *geometry_columns* para todo el servidor. por lo que el f_table_catalog es utilizado para nombrar la base de datos. Postgres, en cambio, guarda una tabla por cada base de datos, así que éste campo está vacío.

El campo coord_dimension es utilizado para saber cuántas dimensiones son utilizadas en la medida de la posición. Un valor del campo de 2 significa que la posición es guardada utilizando las coordenadas (x,y) y un valor de 3 es utilizando las coordenadas (x,y,z).

El tipo de geometría que se puede utilizar depende del objeto que se guarde. Existen tres tipos diferentes o primitivos: Punto, Linea y Polígono.

2. **Spatial_Ref_Sys:** En el tabla 4.2 se muestran los atributos que ésta tabla debe contener junto con una breve explicación de cada uno.

Atributo	**Descripción**
srid	El número de referencia espacial definido por los estándares de la OGC

Continua siguiente página

La solución fue crear una tabla con las diferencias entre cada escala y la información necesaria para poder convertir de una escala a otra sin una pérdida significante de información.

Tabla 4.2 – continuación

Atributo	Descripción
auth_name	El nombre EPSG que identifica el srid
auth_srid	El srid como es definido por el EPGS, el cual generalmente es el mismo que el definido por los estándares OGC.
srtext	El texto de definición utilizado para poner en un mapa la diferencia espacial en formato Projcs.[6]
proj4text	El texto de definición utilizado para poner en un mapa la diferencia espacial en formato Proj4[7]

Tabla 4.2: Atributos de la tabla spatial_ref_sys

Todo dentro de esta tabla son sólo enteros y cadenas. Los atributos srtext y proj4text tienen diferentes significados dependiendo del programa que las esté leyendo. Estos guardan las proyecciones que se utilizan para la información: elíptica, esférica, cónica, etc, que le permite a los desarrolladores y programas transformar de un sistema de coordenadas a otro.

Toda la información geográfica que es guardada en la base de datos tiene que se publicada en algún momento. Para ello se crean estándares como GTFS que permite publicar información relacionada con el transporte público y que gracias a que el Gobierno de la Ciudad de México, así como otros gobiernos e instituciones a nivel mundial, ya lo están utilizando [20] ha sido elegido para el proyecto propuesto en este trabajo de tesis.

4.2. Introducción al estándar GTFS

El estándar GTFS propone un formato común para los horarios de transporte público y la información geográfica asociada a ellos. Toda esta información es manejada en archivos de texto llamados feeds.

Los feeds GTFS permiten que todas las organizaciones de transporte público, tanto públicas como privadas, puedan publicar sus datos de transporte en internet para que programadores e investigadores diseñen aplicaciones que consuman esos datos y mejoren el transporte público.

Un feed se compone de una serie de archivos de texto que son comprimidos dentro de un archivo ZIP. Cada archivo debe modelar un aspecto distinto de la información:

1. Agencias

2. Paradas

[6]Projcs: Es una librería diseñada para realizar conversiones entre proyecciones cartográficas
[7]Proj4: Es una librería diseñada para realizar conversiones entre proyecciones cartográficas

3. Rutas
4. Viajes
5. Tiempos entre paradas
6. Horarios
7. Calendarios
8. Formas de pago
9. Geometría de la ruta
10. Transbordos
11. Información general de la agencia
12. Frecuencias

Esta información se puede utilizar en aplicaciones que permitan ver horarios de los vehículos, la posición de las paradas, analizar posibles rutas alternativas para ir entre dos puntos, entre otras.

Muchas aplicaciones actualmente son compatibles con el formato GTFS, y el método más sencillo para que un archivo sea público es ponerlo dentro de un servidor y permitir descargar el archivo. Los datos con formato GTFS son utilizados prácticamente en todas las aplicaciones de tránsito, planificación de viajes actualmente y buscadores basados en información geográfica. Proyectos como City-Go-Round[8] demuestran las ventajas que existen cuando se tienen datos abiertos que permiten a las personas crear aplicaciones de transporte.

Una ventaja del formato GTFS es que se comienza con información desagregada (sólo la información de arribo y partida en cada parada de cada autobús) y se categoriza la información utilizando un sistema muy parecido a una base de datos con campos y reglas para conectar archivos como si de llaves primarias y llaves foráneas fuesen. Información así de desagregada puede ser utilizada para sistemas de planificación de viajes que te pueden decir a qué hora salir de tu casa para tomar el autobús.

4.3. Modelado del sistema GTFS en la base de datos

Para modelar el estándar GTFS y hacer más sencillo la traducción desde bases de datos a archivo de texto, la base de datos se modeló de la siguiente manera.

[8]http://www.citygoround.org/

4.3.1. Agencia (Agency)

Define el modelo que genera el archivo agency.txt de la especificación. Este es un archivo necesario según los documentos de referencia. [19]

La base de datos maneja una tabla independiente para este archivo con los siguientes atributos en ella:

1. **agency_id (opcional):** El campo agency_id guarda un identificador único para cada una de las agencias de tránsito. Este campo es opcional para archivos que sólo contienen una agencia.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** Sí

   d) *Puede ser blanco:* Sí

2. **agency_name (necesario):** Contiene el nombre completo de la agencia que administra el transporte. Este nombre es el que se desplegará en Google maps.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

3. **agency_url (necesario):** Contiene la URL de la agencia de tránsito. El valor debe ser una URL completa incluyendo el protocolo (http:// o https://), cualquier letra especial debe de escaparse. Las recomendaciones se basan en las establecidas por la w3c. [35]

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

4. **agency_timezone (necesario):** Contiene la zona horaria donde la agencia está establecida. En el caso de este proyecto se utilizará 'America/Mexico_City' basado en la lista de valores válidos. [36]

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

d) *Puede ser blanco:* No

5. **agency_lang (opcional):** Contiene un código de 2 letras siguiendo el estándar ISO 639-1 [37] para indicar el idioma principal que utiliza la agencia. Las mayúsculas y minúsculas son tratadas igual (*ES* o *es* son consideradas iguales).

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 2

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* Sí

6. **agency_phone (opcional):** Contiene un número de teléfono que utiliza la agencia. Este campo debe de ser una cadena. Puede contener marcas de puntuación para agrupar los números. No debe de contener ningún tipo de descripción.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* Sí

7. **agency_fare_url (opcional):** Este campo guarda la URL de la página web en la que es posible comprar boletos de viaje en linea. El valor debe ser una URL completa incluyendo el protocolo (http:// o https://); cualquier letra especial debe de escaparse. Las recomendaciones se basan en las establecidas por la w3c. [35]

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* Sí

En la figura 4.1 se muestra el modelo UML de esta tabla junto con sus relaciones

4.3.2. Tarifas (Fare)

Define el modelo que genera el archivo fares.txt de la especificación. Este es un archivo opcional según los documentos de referencia. [19]

La base de datos maneja una tabla independiente para este archivo con los siguientes atributos en ella:

1. **fare_id (necesario):** Contiene un identificador único para cada uno de los tipos de pago (boleto, tarjeta, efectivo, etc).

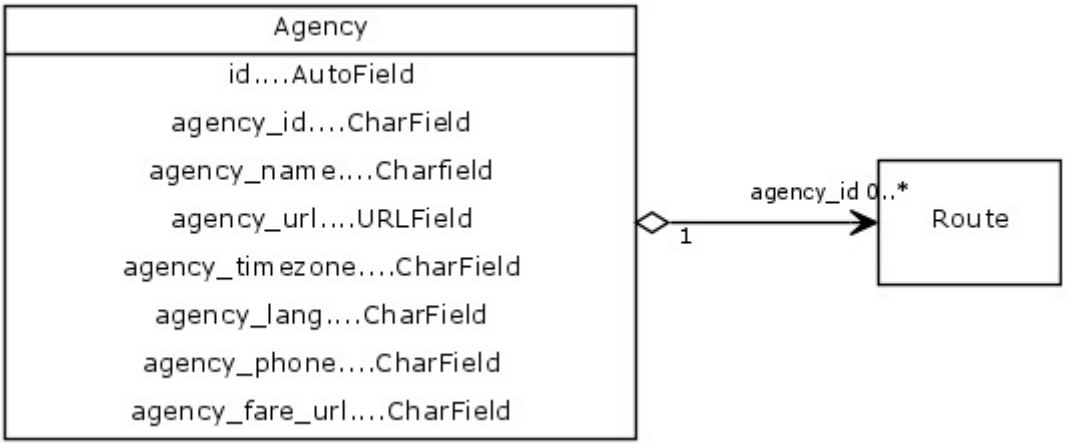

Figura 4.1: UML de la tabla Agency de la base de datos

a) *Tipo de campo:* Cadena

b) ***Longitud máxima:*** 255

c) ***Utilizado como índice:*** Sí

d) *Puede ser blanco:* No

2. **price (necesario):** Contiene el costo del viaje, las unidades son especificadas por el campo currency_type.

   a) *Tipo de campo:* Numérico

   b) ***Longitud máxima:*** 17

   c) ***Número de decimales:*** 4

   d) ***Utilizado como índice:*** No

   e) *Puede ser blanco:* No

3. **currency_type (necesario):** Define la moneda con la que se paga el viaje. Es recomendado utilizar los códigos establecidos en el estándar ISO 4217. [33]

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 3

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

4. **payment_method (necesario):** Este campo guarda el método de pago con el que se debe pagar el viaje. Los valores válidos para este campo son:

   a) 0 - El viaje es pagado en el vehículo (Transporte Concesionado, Taxis, etc).

   b) 1 - El viaje es pagado antes de abordar el transporte (Metro, Metrobús, etc)

a) *Tipo de campo:* Numérico

b) ***Longitud máxima:*** 1

c) ***Utilizado como índice:*** No

d) *Puede ser blanco:* No

5. **transfers (necesario):** Este campo especifica el número de veces que se pueden hacer transbordos entre diferentes vehículos dentro de un sistema de transporte. Los valores válidos para este campo son:

   a) 0 - No se pueden realizar transbordos con un viaje pagado (Taxis)

   b) 1 - Los usuarios se pueden transbordar una vez dentro del sistema

   c) 2 - Los usuarios se pueden transbordar dos veces dentro del sistema

   d) (vacío) - Si el campo está vacío implica que puede haber ilimitados transbordos (Metro y Metrobús están dentro de esta categoría).

   a) *Tipo de campo:* Numérico

   b) ***Longitud máxima:*** 1

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

6. **transfer_duration (opcional):** Especifica el tiempo límite que puede existir entre transbordos antes de que el viaje pagado pierda validez.

   Cuando es utilizado con el campo 'transfers' en 0, representa cuanto tiempo es válido un boleto o viaje. Si el viaje no tiene una duración limitada, este campo debe de ser omitido.

   a) *Tipo de campo:* Numérico

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* Sí

En la figura 4.2 se muestra el modelo UML de la tabla junto con sus relaciones

4.3.3. Reglas de tarifas (Fare rules)

Define el modelo que genera el archivo fare_rules.txt de la especificación. En esta tabla se especifican todas las reglas que se manejan cuando se paga un tipo de transporte. Las reglas pueden ser una o una combinación de las siguientes subreglas:

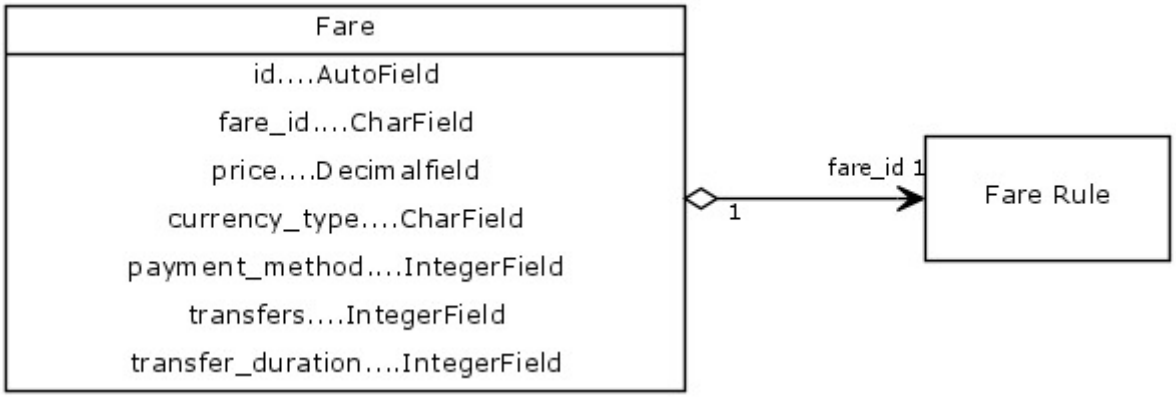

Figura 4.2: UML de la tabla Fare de la base de datos

I. La tarifa depende de las estaciones origen y destino (Taxis, Transporte Concesionado, etc.)

II. La tarifa depende de las zonas que atraviesa el transporte

III. La tarifa depende de la ruta que utiliza el transporte

Este es un archivo opcional según los documentos de referencia. [19]

La base de datos maneja una tabla independiente para este archivo con los siguientes atributos en ella:

1. **fare_id (necesario):** Identificador único para el objeto del tipo de pago.

   a) *Tipo de campo:* Numérico (Llave foránea)

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* No

2. **route_id (opcional):** Asocia el ID de tarifa con una ruta. Los ID de ruta se encuentra en la tabla routes. Si se tienen varias rutas con los mismos reglas de tarifas, se crea una tupla en fare_rules para cada ruta.

   Por ejemplo, si la clase tarifaria p es válida en la ruta TSO y TSE, la tabla fare_rules contendrá las siguientes tuplas:

   p, TSO
   p, TSE

   a) *Tipo de campo:* Numérico (Llave foránea)

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* Sí

3. **origin_id (opcional):** Asocia el ID de tarifa con un zone ID de origen. Los ID de zona se encuentran en la tabla stops. Si se tienen varios ID de origen con los mismos atributos tarifarios se debe crear una tupla en fare_rules para cada ID de origen.

   Por ejemplo, si la clase tarifaria p es válida para todos los viajes que partan desde la zona 2 o la zona 8, la tabla fare_rules contendrá las siguientes tuplas correspondientes:

   p, 2
   p, 8

   a) *Tipo de campo:* Numérico (Llave foránea)

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* Sí

4. **destination_id (opcional):** Asocia el ID de tarifa con un zone ID de destino. Los ID de zona se encuentran en la tabla stops. Si tienes varios ID de destino con los mismos atributos tarifarios, se debe crear una tupla en fare_rules para cada ID de destino.

   Por ejemplo, se puede combinar los campos origin_id y destination_id para especificar que la clase tarifaria p es válida para viajar entre las zonas 3 y 4 y para los viajes entre las zonas 3 y 5, la tabla fare_rules contendría las siguientes tuplas correspondientes a la clase tarifaria:

   p, 3,4
   p, 3,5

   a) *Tipo de campo:* Numérico (Llave foránea)

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* Sí

5. **contains_id (opcional):** Asocia una zona de pago con una zona en específico a través de sus identificadores. La forma de pago, es asociada a su vez con los itinerarios que existen en cada zona.

   Por ejemplo, si la forma de pago p es asociada con la ruta GRT que pasa a través de las zonas 5, 6, 7, entonces la tabla fare_rules debe de contener:

   p,GRT,5
   p,GRT,6
   p,GRT,7

   a) *Tipo de campo:* Numérico (Llave foránea)

b) ***Utilizado como índice:*** No

c) *Puede ser blanco:* Sí

En la figura 4.3 se muestra el modelo UML de esta tabla junto con sus relaciones

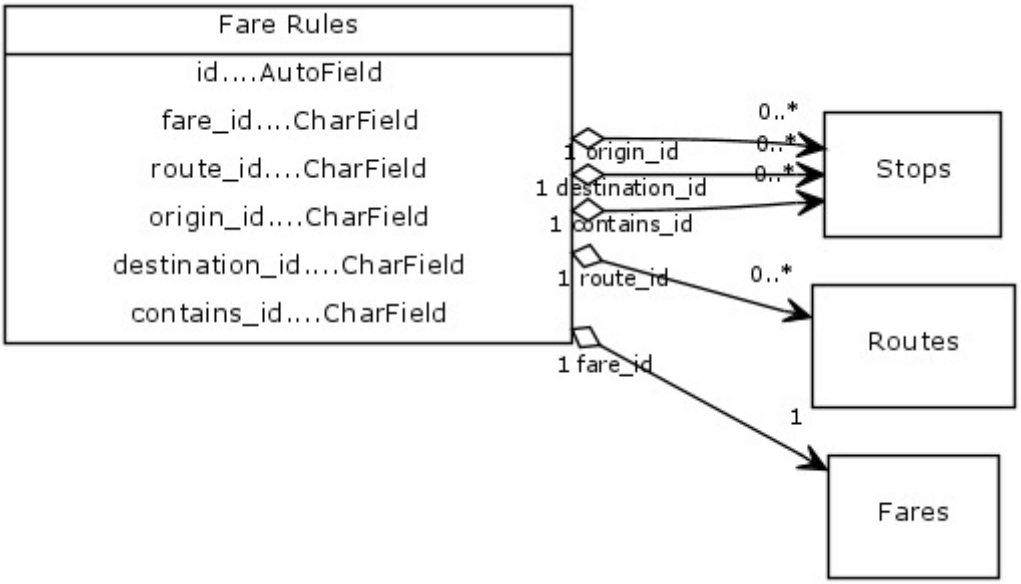

Figura 4.3: UML de la tabla Fare Rules de la base de datos

4.3.4. Información del feed (Feed info)

Define el modelo que genera el archivo feed_info.txt de la especificación.

La tabla contiene información sobre el feed mismo en lugar de los servicios que describe el feed. Actualmente el estándar GTFS maneja el archivo agency para brindar información sobre las empresas que operan los servicios descritos en el feed. El administrador del feed puede ser una entidad diferente de cualquiera de las empresas (en el caso de los administradores regionales) que proveen servicios.

Este es un archivo necesario según los documentos de referencia. [19]

La base de datos maneja una tabla independiente para este archivo con los siguientes atributos en ella:

1. **feed_publisher_name (necesario):** Contiene el nombre completo de la organización que administra y publica el archivo. Este puede ser el mismo que el nombre de la agencia en la tabla agency.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

2. **feed_publisher_url (necesario)** Contiene la url de la página web de la organización que publica la información. Este puede ser el mismo que la url de la agencia en la tabla agency y debe seguir las mismas reglas.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

3. **feed_lang (necesario):** Campo que contiene un código basado en el estándar IETF BCP 47 [38], el cual especifica el lenguaje utilizado en este feed. Esto ayuda a los usuarios a manejar la información y realizar traducciones.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 20

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

4. **feed_start_date (opcional):**

5. **feed_end_date (opcional):** Estos campos manejan el tiempo abarcado por la información que se está enviando. El formato para las fechas es YYYYDDMM. El campo feed_end_date no debe ser anterior que el campo feed_start_date.

   a) *Tipo de campo:* Fecha o Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* Sí

6. **feed_version (opcional):** Este campo especifica la versión de la información que se esta publicando. Esto es utilizado por programadores para saber si están trabajando con la última versión de la información. El valor que puede obtener este campo no tiene que ser un número solamente, puede seguir las convenciones de programación en numeración de versiones. [39]

1. *Tipo de campo:* Cadena

2. ***Longitud máxima:*** 20

3. ***Utilizado como índice:*** No

4. *Puede ser blanco:* Sí

En la figura 4.4 se muestra el modelo UML de esta tabla junto con sus relaciones

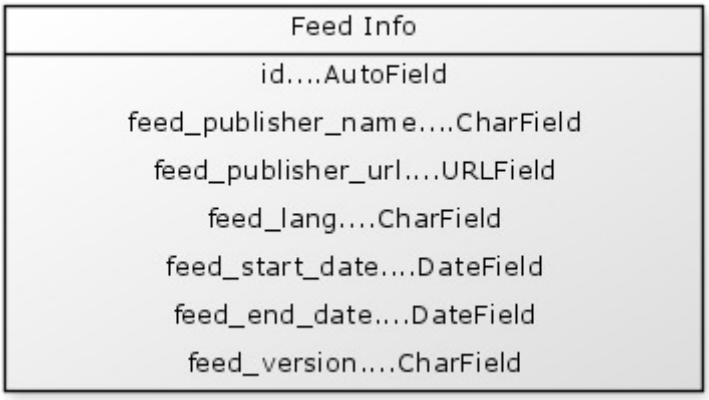

Figura 4.4: UML de la tabla Feed Info de la base de datos

4.3.5. Frecuencias (frequencies)

Define el modelo que genera el archivo frequencies.txt de la especificación.

Este modelo representa horarios que no tienen una lista de paradas fijas. Esto implica que los servicios que se insertan en este modelo no tienen un tiempo de arribo o un tiempo de partida definidos. En lugar, la tabla stop_times define la secuencia de paradas y la diferencia entre cada ellas. Este modelo es ideal para Taxis y Transporte Concesionado en la zona metropolitana. Este es un archivo opcional según los documentos de referencia. [19]

La base de datos maneja una tabla independiente para este archivo con los siguientes atributos en ella:

1. **trip_id (necesario):** Identifica el viaje en el cual la frecuencia aplica. Este ID es una llave foránea de la tabla trips.

   a) *Tipo de campo:* Numérico (Llave foránea)

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* No

2. **start_time (necesario):** El campo start_time especifica la hora a la que empieza el servicio con la frecuencia especificada. La hora se calcula como mediodía menos 12 h (lo que corresponde a la medianoche, excepto durante el período en el que se aplica el cambio de horario de verano/invierno) al principio de la fecha de servicio. En el caso de las horas posteriores a la medianoche, la hora se introduce como un valor mayor que 24:00:00 en la hora local HH:MM:SS en el día en que empieza el horario del viaje. Por ejemplo, 25:35:00.

   a) *Tipo de campo:* Fecha o Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

3. **end_time (necesario):** El campo end_time especifica la hora a la que finaliza el servicio con la frecuencia especificada. La hora se calcula como mediodía menos 12 h (lo que corresponde a la medianoche, excepto durante el período en el que se aplica el cambio de horario de verano/invierno) al principio de la fecha de servicio. En el caso de las horas posteriores a la medianoche, la hora se introduce como un valor mayor que 24:00:00 en la hora local HH:MM:SS en el día en que empieza el horario del viaje. Por ejemplo, 25:35:00.

   El valor de end_time no debe ser menor que el valor de start_time

   a) *Tipo de campo:* Fecha

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

4. **headway_secs (necesario):** Indica el período de tiempo entre salidas desde la misma parada (tiempo entre viajes) para este tipo de viaje durante el intervalo de tiempo especificado mediante start_time y end_time. El valor correspondiente al tiempo entre viajes debe introducirse en segundos.

   Los períodos en los que se definen los tiempos entre viajes (las tuplas de frequencies) no se deben solapar para un mismo viaje ya que resultaría difícil determinar el significado de dos tiempos solapados. Sin embargo, un período de tiempo entre viajes puede comenzar en el mismo momento en que acabe otro, por ejemplo:

   a) A, 05:00:00, 07:00:00, 600

   b) B, 07:00:00, 12:00:00, 1200

   a) *Tipo de campo:* Numérico

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

5. **exact_times (opcional):** Determina si los viajes basados en frecuencias deben estar programados de manera exacta según la información especificada de tiempo entre viajes. Los posibles valores válidos para este campo son:

   I. 0 o (vacío): los viajes basados en frecuencias no están programados de manera exacta. Este es el comportamiento predeterminado.

   II. 1: los viajes basados en frecuencias están programados de manera exacta. En una fila de frequencies, los viajes se programan a partir de

$$trip\_start\_time = start\_time + x * headway\_secs \; \forall x \in (0, 1, 2, ...)$$

$$trip\_start\_time < end\_time$$

El valor de exact_times debe ser el mismo para todas las tuplas de "frequencies" con el mismo trip_id. Si el valor de exact_times es 1 y una fila de frequencies tiene un valor de start_time igual a end_time, no se debe programar ningún viaje. Si el valor de exact_times es 1, se debe tener cuidado de elegir un valor de end_time mayor que la hora de inicio del último viaje deseado, pero menor que la hora de inicio del último viaje deseado + headway_secs.

a) *Tipo de campo:* Cadena

b) ***Longitud máxima:*** 1

c) ***Utilizado como índice:*** No

d) *Puede ser blanco:* Sí

En la figura 4.5 se muestra el modelo UML de esta tabla junto con sus relaciones

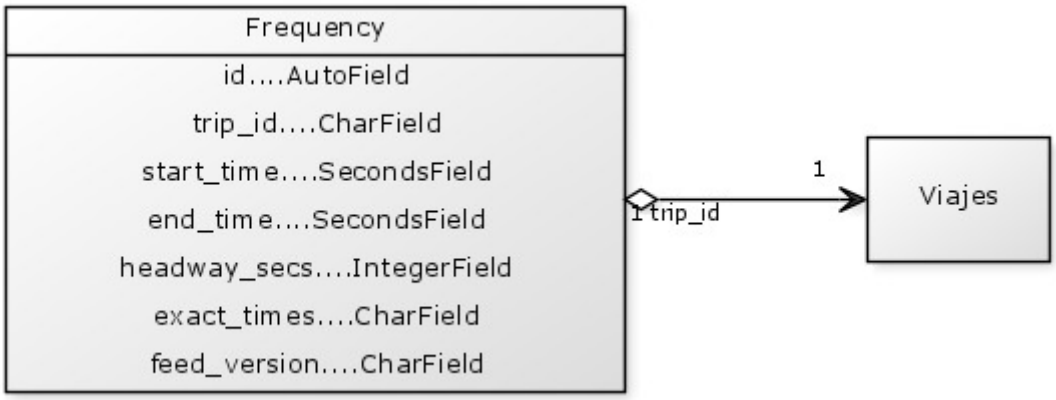

Figura 4.5: UML de la tabla Frecuencies de la base de datos

4.3.6. Rutas (Route)

Define el modelo que genera el archivo routes.txt de la especificación. Este es un archivo necesario según los documentos de referencia. [19]

La base de datos maneja una tabla independiente para este archivo con los siguientes atributos en ella:

1. **route_id (necesario):** El campo route_id es un identificador único para la ruta. Los identificadores deben de ser palabras únicas sin repetir.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** Sí

d) *Puede ser blanco:* No

2. **agency_id (opcional):** El campo agency_id identifica una empresa para la ruta especificada. Este valor se menciona en la tabla agency. Es utilizado cuando se proporciona datos de rutas correspondientes a más de una empresa.

   a) *Tipo de campo:* Numérico (Llave foránea)

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* Sí

3. **route_short_name (necesario):** Contiene el nombre corto de una ruta. Suele ser un identificador corto y abstracto como 32, 100X o Verde que los usuarios usan para identificar la ruta pero que no proporciona ninguna indicación sobre los puntos que atraviesa. Si no tiene un nombre corto se debe utilizar una cadena vacía como valor para este campo.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 10

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

4. **route_long_name (necesario):** Contiene el nombre completo de una ruta. Este nombre suele ser más descriptivo que el route_short_name y suele incluir el destino o parada de la ruta. Si no tiene un nombre largo de debe utilizar una cadena vacía como valor para este campo.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

5. **route_desc (opcional):** Contiene una descripción de una ruta. Es utilizado para dar información más completa sobre la ruta que se está eligiendo.

   a) *Tipo de campo:* Texto

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* Sí

6. **route_type (necesario):** Describe el tipo de transporte público utilizado en una ruta. Los valores válidos para este campo son:

I. 0: tranvía, tren ligero. Cualquier metro ligero o sistema ferroviario en superficie dentro de un área metropolitana.

II. 1: subterráneo, metro. Cualquier sistema ferroviario subterráneo dentro de un área metropolitana.

III. 2: tren. Utilizado para viajes de larga distancia.

IV. 3: autobús. Utilizado para rutas en autobús de corta y larga distancia.

V. 4: transbordador, ferry. Utilizado para servicio de transporte público fluvial o marítimo de corta y larga distancia.

VI. 5: funicular. Utilizado para funiculares en superficie en donde el cable pasa por debajo del vehículo.

VII. 6: cabina, vehículo suspendido de un cable. Normalmente se utiliza para medios de transporte público por cable en donde el vehículo queda suspendido de un cable.

VIII. 7: funicular. Cualquier sistema diseñado para recorridos con una gran inclinación.

a) *Tipo de campo:* Numérico

b) ***Longitud máxima:*** 1

c) ***Utilizado como índice:*** No

d) *Puede ser blanco:* No

7. **route_url :** Contiene la URL de una página web relativa a la ruta concreta. El valor debe ser una URL completa incluyendo el protocolo (http:// o https://), cualquier letra especial debe de escaparse. Las recomendaciones se basan en las establecidas por la w3c. [35]

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

8. **route_color (opcional):**

   Para aquellos sistemas que tienen colores para identificar las rutas define su color correspondiente. Este color se debe proporcionar en formato hexadecimal como, por ejemplo, 00FFFF. Si no se especifica ningún color, el color de ruta predeterminado es el blanco (FFFFFF).

   El contraste entre los colores de route_color y route_text_color debe ser suficiente como para distinguirlos en una pantalla en blanco y negro. [40]. Existen herramientas en línea útiles para elegir colores que contrasten como:

http://snook.ca/technical/colour_contrast/colour.html

a) *Tipo de campo:* Cadena

b) ***Longitud máxima:*** 6

c) ***Utilizado como índice:*** No

d) *Puede ser blanco:* Sí

9. **route_text_color (opcional):** Se puede usar para especificar un color legible para el texto incluido sobre un fondo del valor route_color. Este color se debe proporcionar en formato hexadecimal como, por ejemplo, FFD700. Si no se especifica ningún tipo de ubicación, el color del texto predeterminado es el negro (000000).

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 6

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* Sí

En la figura 4.6 se muestra el modelo UML de esta tabla junto con sus relaciones

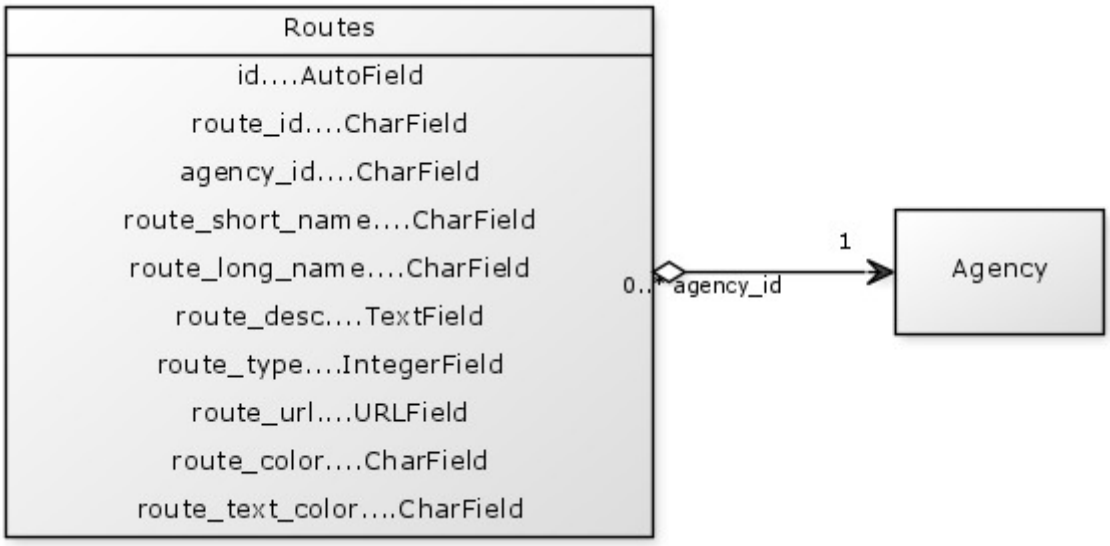

Figura 4.6: UML de la tabla Routes de la base de datos

4.3.7. Calendario (Calendar)

Define el modelo que genera el archivo calendar.txt de la especificación. Este campo tiene un patrón que se repite a lo largo de cada una de las tuplas que lo componen. Cada uno de los campos representa un día de la semana y manejan los siguientes valores

1. Un valor de 1 indica que el servicio está disponible ése día que representa y es válido dentro del intervalo de fechas. Por ejemplo, si el campo es 1 en wednesday, start_date es 20130912 y end_date es 20130926 todos los miércoles entre el 12 de septiembre y el 26 de septiembre el servicio estará activo. (El intervalo de fechas se especifica mediante los campos start_date y end_date).

2. Un valor de 0 indica que el servicio no está disponible ese día y puede es válido dentro del intervalo de fechas.

Debido a que los campos dentro del archivo se llaman service y no calendar, la clase que maneja este campo se ha decidido llamar Service. Este es un archivo necesario según los documentos de referencia. [19]

La base de datos maneja una tabla independiente para este archivo con los siguientes atributos en ella:

1. **service_id (necesario):** Contiene un ID que identifica de forma exclusiva un conjunto de fechas en el que el servicio está disponible en una o más rutas. Cada valor de service_id puede aparecer como máximo una vez en un archivo calendar.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** Sí

   d) *Puede ser blanco:* No

2. **start_date (necesario):** Contiene la fecha de inicio del servicio, debe tener el formato AAAAMMDD.

   a) *Tipo de campo:* Fecha

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* No

3. **end_date (necesario):** Contiene la fecha de finalización del servicio. Esta fecha se incluye en el intervalo del servicio.

   El valor del campo end_date debe tener el formato AAAAMMDD.

   a) *Tipo de campo:* Fecha

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* No

4. **monday...sunday (necesario):** Representan una matriz de días de la semana para los cuales se especifica si el servicio está activo o no.

a) *Tipo de campo:* Booleano

b) ***Utilizado como índice:*** No

c) *Puede ser blanco:* No

En la figura 4.7 se muestra el modelo UML de esta tabla junto con sus relaciones

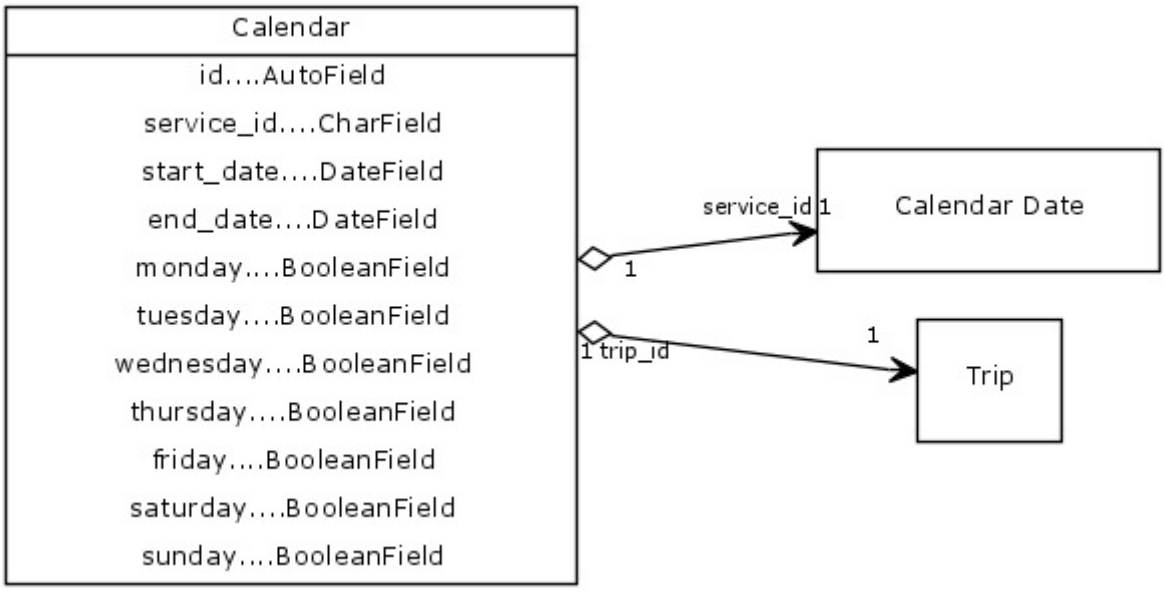

Figura 4.7: UML de la tabla Calendar de la base de datos

4.3.8. Fechas de Calendario (Calendar Date)

Define el modelo que genera el archivo calendar_dates.txt de la especificación.

La tabla calendar_dates permite habilitar o inhabilitar de forma explícita una fecha en concreto en el calendario identificado por service_id.

Este es un archivo opcional según los documentos de referencia. [19] La base de datos maneja una tabla independiente para este archivo con los siguientes atributos en ella:

1. **service_id (necesario):** Contiene un ID que identifica de forma exclusiva un conjunto de fechas en la tabla calendar. Cada par (service_id, date) solo puede aparecer una vez en calendar_dates. Si el valor service_id aparece tanto en la tabla calendar como en calendar_dates, la información de calendar_dates modifica la información del servicio especificada en calendar.

   a) *Tipo de campo:* Numérico (Llave foránea)

   b) ***Utilizado como índice:*** Sí

   c) *Puede ser blanco:* No

2. **date (necesario):** Especifica una fecha concreta en la que la disponibilidad del servicio sea diferente a la de la norma. Es posible utilizar el campo exception_type para indicar cuál es la situación de los servicios en éste día.

   El valor del campo date debe tener el formato AAAAMMDD.

   a) *Tipo de campo:* Fecha o Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

3. **exception_type (necesario):** Indica si el servicio está o no disponible en la fecha especificada en el campo date. Los valores que puede tener éste campo son:

   I. 1: Indica que el servicio se ha agregado a la fecha especificada.

   II. 2: Indica que el servicio se ha suprimido en la fecha especificada.

   Por ejemplo, suponiendo que una ruta tiene un conjunto de viajes disponibles durante los días feriados y otro conjunto de viajes disponibles el resto de días. Es posible tener un service_id que corresponda al horario de servicio regular y otro service_id que corresponda al horario de días feriados. En el caso de un feriado concreto se puede utilizar la tabla calendar_dates para agregar el feriado al service_id de días feriados y eliminar dicho feriado del horario de service_id regular.

En la figura 4.8 se muestra el modelo UML de esta tabla junto con sus relaciones

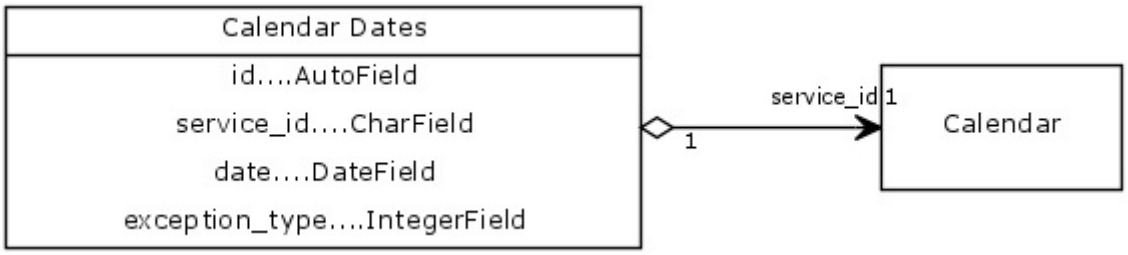

Figura 4.8: UML de la tabla Calendar Dates de la base de datos

4.3.9. Formas de ruta (Shapes)

Define el modelo que genera el archivo shapes.txt de la especificación. Este es un archivo opcional según los documentos de referencia. [19]

La base de datos maneja dos tablas relacionadas *Shape* y *Shape Point* para representar éste archivo con los siguientes atributos en ella:

4.3.9.1. Tabla Shapes

1. **shape_id (required):** Es un ID que identifica exclusivamente un tipo de forma para una ruta

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** Sí

   d) *Puede ser blanco:* No

4.3.9.2. Tabla Shape Point

1. **shape_pt_lat (required):** El campo shape_pt_lat asocia la latitud de un punto de una forma con un ID de forma. El valor de este campo debe ser una latitud WGS 84 [41] válida. Cada fila de la tabla shapes representa un punto de la forma en la definición de dicha forma.

   Por ejemplo, si la forma A_shp tiene tres puntos en su definición, la tabla shapes puede contener las tuplas siguientes para definir la forma:

   A_shp,37.61956, 15.48161,0
   A_shp,37.64430, 15.41070,6
   A_shp,37.65863, 15.30839,11

   a) *Tipo de campo:* Numérico

   b) ***Longitud máxima:*** 13

   c) ***Número de decimales:*** 8

   d) ***Utilizado como índice:*** No

   e) *Puede ser blanco:* No

2. **shape_pt_lon (required):** El campo shape_pt_lon asocia la longitud de un punto de una forma con una forma. El valor de este campo debe ser una longitud WGS 84 [41] válido de -180 a 180. Cada fila de la tabla shapes representa un punto de la forma en la definición de dicha forma.

   Por ejemplo, si la forma A_shp tiene tres puntos en su definición, la tabla shapes puede contener las tuplas siguientes para definir la forma:

   A_shp,37.61956,-122.48161,0
   A_shp,37.64430,-122.41070,6
   A_shp,37.65863,-122.30839,11

a) *Tipo de campo:* Numérico

b) ***Longitud máxima:*** 13

c) ***Número de decimales:*** 8

d) ***Utilizado como índice:*** No

e) *Puede ser blanco:* No

3. **shape_pt_sequence (required):** El campo shape_pt_sequence asocia la latitud y la longitud de un punto de una forma al orden secuencial que tienen a lo largo de la forma. Los valores de shape_pt_sequence deben ser enteros no negativos y deben aumentar a lo largo del viaje.

   Por ejemplo, si la forma A_shp tiene tres puntos en su definición, la tabla shapes puede contener las tuplas siguientes para definir la forma:

   A_shp,37.61956,-122.48161,0
   A_shp,37.64430,-122.41070,1
   A_shp,37.65863,-122.30839,2

   a) *Tipo de campo:* Numérico

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* No

4. **shape_dist_traveled (opcional):** Cuando se utiliza en la tabla shapes, éste campo posiciona un punto en la ruta como la distancia recorrida a lo largo de una forma desde el primer punto. Representa una distancia real recorrida a lo largo de la ruta expresada en unidades como pies o kilómetros. Esta información permite que quien planifica el viaje determine la porción de la forma que se debe trazar al mostrar parte de un viaje en el mapa. Los valores usados para shape_dist_traveled deben aumentar junto con shape_pt_sequence: no se pueden usar para mostrar el recorrido inverso a lo largo de una ruta.

   Las unidades utilizadas para shape_dist_traveled en la tabla shapes deben coincidir con las unidades que se utilizan para este campo en la tabla stop_times.

   Por ejemplo, si un autobús atraviesa los tres puntos definidos anteriormente para A_shp, los valores adicionales de shape_dist_traveled (mostrados aquí en kilómetros) tendrán la apariencia siguiente:

   A_shp,37.61956,-122.48161,0,0
   A_shp,37.64430,-122.41070,1,6.8310
   A_shp,37.65863,-122.30839,2,15.8765

a) *Tipo de campo:* Numérico

b) ***Utilizado como índice:*** No

c) *Puede ser blanco:* No

En la figura 4.9 se muestra el modelo UML de esta tabla junto con sus relaciones

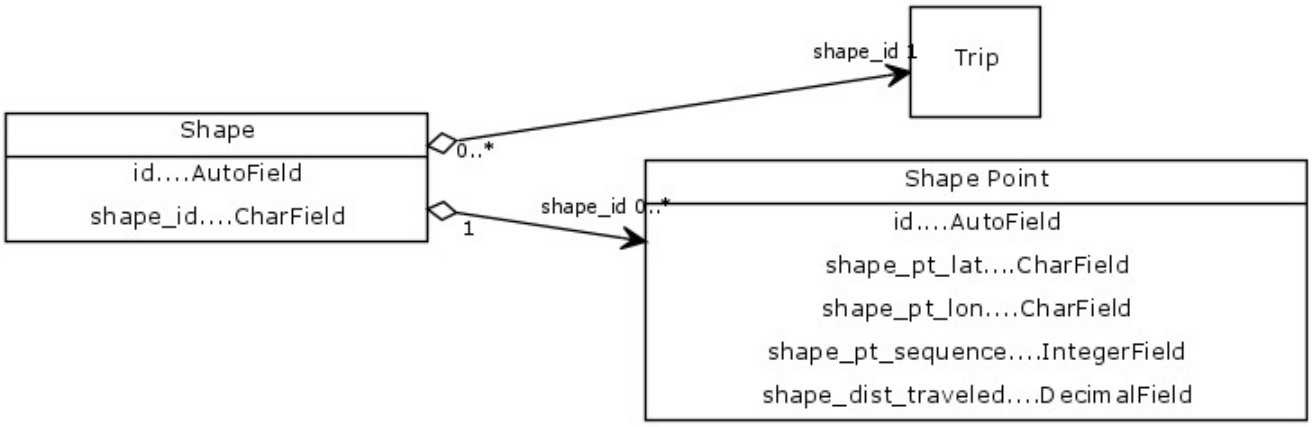

Figura 4.9: UML de la tabla Shapes de la base de datos

4.3.10. Paradas (Stops)

Define el modelo que genera el archivo stops.txt de la especificación. Este es un archivo necesario según los documentos de referencia. [19]

La base de datos maneja una tabla independiente para este archivo con los siguientes atributos en ella:

1. **stop_id (necesario):** Es un ID que identifica de forma exclusiva a una parada o estación. Es posible que varias rutas utilicen una misma parada.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** Sí

   d) *Puede ser blanco:* No

2. **stop_code (opcional):** Contiene texto corto o un número que identifica de forma exclusiva la parada de los pasajeros. Los códigos de parada se suelen usar en sistemas de información sobre transporte público para teléfonos o impresos en los carteles de paradas para facilitar a los usuarios la consulta de los horarios de parada o información en tiempo real sobre llegadas de un vehículo.

   El campo stop_code se debe usar únicamente para códigos de parada que se muestran a los pasajeros. En los códigos internos, usa stop_id.

   Este campo se debe dejar en blanco en el caso de paradas sin código.

   a) *Tipo de campo:* Cadena
   b) ***Longitud máxima:*** 255
   c) ***Utilizado como índice:*** No
   d) *Puede ser blanco:* Sí

3. **stop_name (necesario):** Contiene el nombre de una parada o estación. Se recomienda usar un nombre que resulte comprensible para la gente local y turistas.

   a) *Tipo de campo:* Cadena
   b) ***Longitud máxima:*** 255
   c) ***Utilizado como índice:*** No
   d) *Puede ser blanco:* No

4. **stop_desc (opcional):** Contiene la descripción de una parada. Está diseñado para incluir información útil.

   a) *Tipo de campo:* Cadena
   b) ***Longitud máxima:*** 255
   c) ***Utilizado como índice:*** No
   d) *Puede ser blanco:* Sí

5. **stop_lat (necesario):** Contiene la latitud de una parada o estación. El valor de este campo debe ser una latitud WGS 84 [41] válida.

   a) *Tipo de campo:* Numérico
   b) ***Longitud máxima:*** 13
   c) ***Número de lugares decimales:*** 8
   d) ***Utilizado como índice:*** No
   e) *Puede ser blanco:* No

6. **stop_lon (necesario):** Contiene la longitud de una parada o estación. El valor de este campo debe ser una latitud WGS 84 [41] válida entre -180 y 180.

   a) *Tipo de campo:* Numérico
   b) ***Longitud máxima:*** 13
   c) ***Número de lugares decimales:*** 8
   d) ***Utilizado como índice:*** No

e) *Puede ser blanco:* No

7. **zone_id (opcional):** Define la zona tarifaria de un ID de parada. Los ID de zona son necesarios si deseas proporcionar información de pasajes mediante fare_rules.

   a) *Tipo de campo:* Numérico (Llave foránea)

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* No

8. **stop_url (opcional):** Contiene la URL de una página web relativa a una parada concreta. El valor de este campo debe ser diferente al de los campos agency_url y route_url.

   El valor debe ser una URL completa incluyendo el protocolo (http:// o https://), cualquier letra especial debe de escaparse. Las recomendaciones se basan en las establecidas por la w3c. [35]

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* Sí

9. **location_type (opcional):** Identifica si este ID de parada representa a una parada o estación. Si no se especifica el tipo de ubicación o si el campo location_type está vacío los ID de parada se consideran como una parada. Es posible que las estaciones tengan propiedades diferentes a las paradas cuando se representan en un mapa o se usan en la planificación de viajes.

   El campo location_type puede tener los valores siguientes:

   I. 0 o en blanco: Parada. Ubicación en la que los pasajeros suben a bordo o bajan de un medio de transporte público.

   II. 1: estación. Estructura física o área que contiene una o más paradas.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 1

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* Sí

10. **parent_station (opcional):** Para las paradas que se encuentran físicamente en el interior de estaciones, el campo parent_station identifica la estación relacionada éstas. Para usar este campo stops también debe incluir una fila en la que se asigne a este ID de parada el tipo de ubicación 1.

a) *Tipo de campo:* Numérico (Llave foránea)

b) ***Utilizado como índice:*** No

c) *Puede ser blanco:* Sí

11. **stop_timezone (opcional):** El campo stop_timezone contiene la zona horaria en la que se ubica esta parada o estación. [36]

    Si una parada tiene una estación de origen, se considerará que la parada está en la zona horaria especificada en el valor stop_timezone de la estación. Si la estación de origen no tiene un valor de stop_timezone se supondrá que las paradas que pertenezcan a esa estación están en la zona horaria especificada por agency_timezone de la tabla agency, incluso si otras paradas tienen sus propios valores de stop_timezone. En otras palabras, si una parada específica tiene un valor de parent_station, cualquier valor de stop_timezone especificado para esa parada debe ignorarse.

    Incluso si los valores de stop_timezone se proporcionan en stops, los horarios en stop_times se deberán continuar especificando como horario desde la medianoche en la zona horaria especificada por la agency_timezone en agency. Esto garantiza que los valores de horarios en un viaje siempre aumenten durante el transcurso de un viaje, independientemente de las zonas horarias que atraviese el viaje.

    a) *Tipo de campo:* Cadena

    b) ***Longitud máxima:*** 255

    c) ***Utilizado como índice:*** No

    d) *Puede ser blanco:* Sí

En la figura 4.10 se muestra el modelo UML de esta tabla junto con sus relaciones

4.3.11. Tiempos entre paradas (Stop times)

Define el modelo que genera el archivo stop_times.txt de la especificación. Este es un archivo necesario según los documentos de referencia. [19]

La base de datos maneja una tabla independiente para este archivo con los siguientes atributos en ella:

1. **trip_id (necesario):** Contiene un ID que identifica un viaje. Este valor se encuentra en la tabla trips.

    a) *Tipo de campo:* Numérico (Llave foránea)

    b) ***Utilizado como índice:*** No

    c) *Puede ser blanco:* No

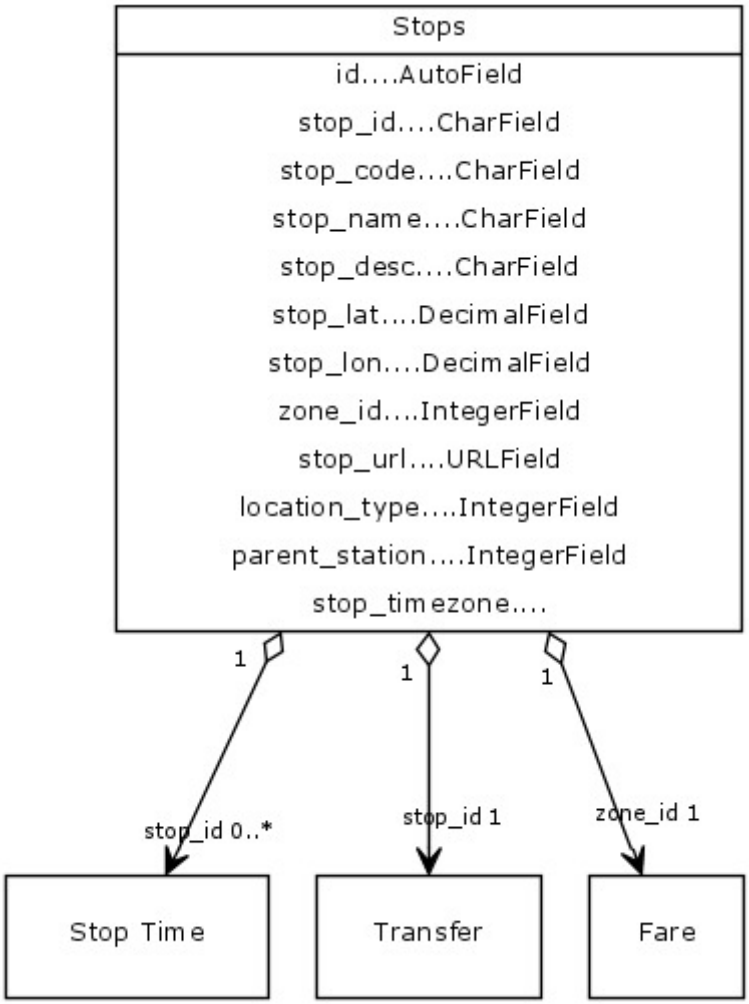

Figura 4.10: UML de la tabla Stops de la base de datos

2. **arrival_time (necesario):** Especifica la hora de llegada a una parada concreta correspondiente a un viaje específico de una ruta. La hora se calcula como mediodía menos 12 h (lo que corresponde a la medianoche, excepto durante el período en el que se aplica el cambio de horario de verano/invierno) al principio de la fecha de servicio. En el caso de las horas posteriores a la medianoche de la fecha de servicio, ingresa la hora como un valor mayor que 24:00:00 en la hora local HH:MM:SS en el día en que empieza el horario del viaje. Si no se dispone de horas diferentes para la llegada y la salida de una parada se debe ingresar el mismo valor para arrival_time y departure_time.

   Se debe especificar la horas de llegada correspondientes a la última parada de un viaje. Si esta parada no es un punto temporal, entonces se puede utilizar un valor de cadena vacía en los campos arrival_time y departure_time. Las paradas sin horas de llegada se programarán de acuerdo con la parada con tiempo anterior que esté más cerca. No se debe interpolar las paradas.

   Las horas deben tener ocho dígitos en formato HH:MM:SS (también se acepta H:MM:SS si la hora empieza por 0).

Hora	Valor arrival_time
08:10:00 A.M.	08:10:00 u 8:10:00
01:05:00 P.M.	13:05:00
07:40:00 P.M.	19:40:00
01:55:00 A.M.	25:55:00

Tabla 4.3: Valores posibles de la propiedad arrival_time

Nota: Los viajes que abarcan varias fechas deben tener horas de parada mayores que 24:00:00. Por ejemplo, si un viaje empieza a las 10:30:00 p. m. y termina a las 2:15:00 a. m. del día siguiente, las horas de parada serían 22:30:00 y 26:15:00.

a) *Tipo de campo:* Cadena

b) ***Utilizado como índice:*** No

c) *Puede ser blanco:* Sí

3. **departure_time (necesario):** Especifica la hora de salida de una parada concreta correspondiente a un viaje específico en una ruta. Las reglas de cálculo de los horarios son idénticas que en el campo arrival_time.

Hora	Valor departure_time
08:10:00 A.M.	08:10:00 u 8:10:00
01:05:00 P.M.	13:05:00
07:40:00 P.M.	19:40:00
01:55:00 A.M.	25:55:00

Tabla 4.4: Valores posibles de la propiedad departure_time

a) *Tipo de campo:* Cadena

b) ***Utilizado como índice:*** No

c) *Puede ser blanco:* Sí

4. **stop_id (necesario):** Es un ID que identifica de forma exclusiva a una parada. Es posible que varias rutas utilicen una misma parada. El stop_id se encuentra en la tabla stops. Si se utiliza location_type, todas las paradas a las que se haga referencia en stop_times deben tener un tipo de ubicación (location_type) igual a 0.

   Siempre que sea posible, debe mantenerse la coherencia de los valores de stop_id al realizar actualizaciones de los feeds. En otras palabras, una parada A con un valor stop_id 1 debe tener un valor stop_id 1 en las actualizaciones de datos posteriores. Si una parada no

es un punto temporal, se deben ingresar valores en blanco en los campos arrival_time y departure_time.

a) *Tipo de campo:* Numérico (Llave foránea)

b) ***Utilizado como índice:*** No

c) *Puede ser blanco:* No

5. **stop_sequence (necesario):** Identifica el orden de las paradas en un viaje en concreto. Los valores de stop_sequence deben ser enteros no negativos y deben aumentar durante el viaje.

   Por ejemplo, la primera parada del viaje podría tener un valor stop_sequence de 1, la segunda un valor stop_sequence de 2; la tercera, un valor stop_sequence de 3 y así sucesivamente.

   a) *Tipo de campo:* Numérico

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* No

6. **stop_headsign (opcional):**

   Contiene el texto que aparece en el cartel que identifica el destino del viaje para los pasajeros. Se puede utilizar este campo para anular el valor predeterminado trip_headsign cuando la señal de destino cambie de una parada a otra.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 255

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* Sí

7. **pickup_type (opcional):** Indica si se recogen o no los pasajeros en una parada. Este campo también permite a la empresa de transporte público indicar que los pasajeros deben llamar a la empresa o notificar al conductor que es necesario recogerlos en esa parada en concreto (compañías de taxis). Los valores válidos para este campo son:

   I. 0: El transporte recoge regularmente en la parada.

   II. 1 : El transporte no recoge en esa parada.

   III. 2 : Se debe llamar a la empresa para pedir por un transporte.

   IV. 3 : Se debe coordinar con los conductores de los transportes para que lo recojan.

   El valor predeterminado de este campo es 0.

a) *Tipo de campo:* Numérico

b) ***Longitud máxima:*** 1

c) ***Utilizado como índice:*** No

d) *Puede ser blanco:* Sí

8. **drop_off_type (opcional):** Indica si los pasajeros se bajan en una parada como parte del horario normal o si no pueden bajar en esa parada. Este campo también permite a la empresa de transporte público indicar que los pasajeros si deben llamar a la empresa o notificar la conductor que deben bajarse en esa parada. Los valores válidos para este campo son:

   I. 0: Parada programada regularmente.

   II. 1: Sin parada disponible.

   III. 2: Se debe llamar a la empresa para organizar la parada.

   IV. 3: Se debe pedir al conductor que pare.

   El valor predeterminado de este campo es 0.

   a) *Tipo de campo:* Numérico

   b) ***Longitud máxima:*** 1

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* Sí

9. **shape_dist_traveled (opcional):** Cuando se utiliza en la tabla stop_times, el campo shape_dist_traveled posiciona una parada como una distancia desde el primer punto de la forma. shape_dist_traveled representa la distancia real recorrida a lo largo de la ruta expresada en unidades como pies o kilómetros. Por ejemplo, si un autobús recorre una distancia de 5.25 kilómetros desde el principio de la ruta hasta la parada, el campo shape_dist_traveled del ID de parada correspondiente se expresaría como 5.25. Esta información permite que quien planifica el viaje determine la porción de la forma que se debe trazar al mostrar parte de un viaje en el mapa. Los valores utilizados para shape_dist_traveled se deben incrementar junto con los de stop_sequence: no se pueden usar para mostrar el recorrido inverso a lo largo de una ruta.

   Las unidades utilizadas para shape_dist_traveled en la tabla stop_times deben coincidir con las unidades que se utilizan para este campo en la tabla shapes.

   a) *Tipo de campo:* Numérico

   b) ***Longitud máxima:*** 10

   c) ***Número de decimales:*** 2

d) ***Utilizado como índice:*** No

e) *Puede ser blanco:* Sí

En la figura 4.11 se muestra el modelo UML de esta tabla junto con sus relaciones

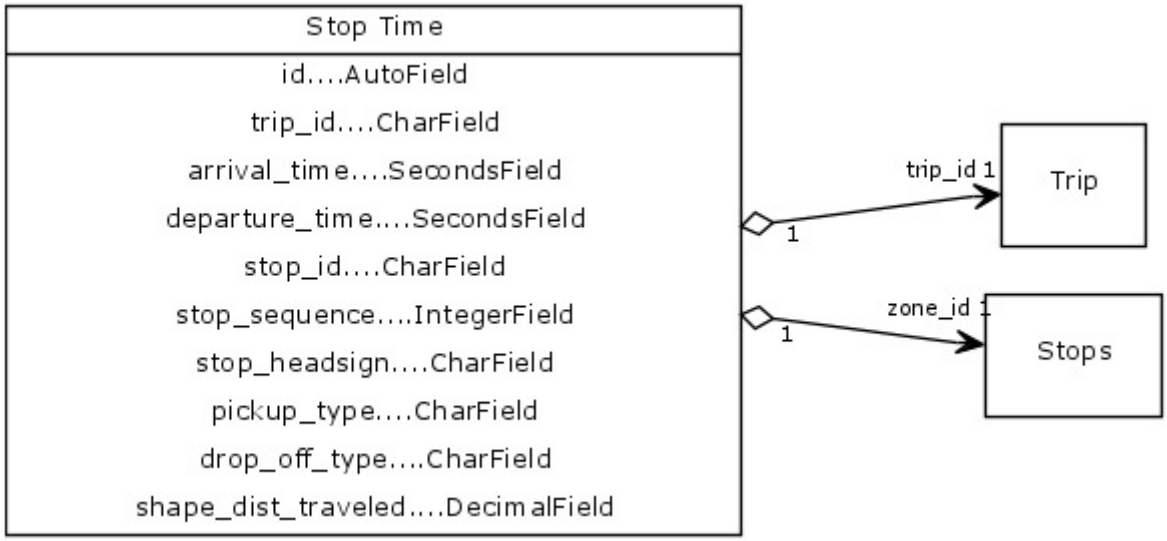

Figura 4.11: UML de la tabla Stops Times de la base de datos

4.3.12. Transbordos (Transfers)

Define el modelo que genera el archivo transfers.txt de la especificación.

Las personas que planifican viajes normalmente calculan los puntos de transbordo según la proximidad relativa de las paradas en cada ruta. En el caso de pares de paradas que pudieran resultar ambiguas o de transbordos para los que se deseé especificar una opción concreta, se puede utilizar la tabla transfers para definir reglas adicionales para realizar conexiones entre rutas. Este es un archivo opcional según los documentos de referencia. [19]

La base de datos maneja una tabla independiente para este archivo con los siguientes atributos en ella:

1. **from_stop_id (necesario):** Contiene un ID que identifica una parada o estación en la que comienza una conexión entre rutas. Los ID se encuentran en la tabla stops. Si el ID de parada hace referencia a una estación que contiene varias paradas, esta regla de transbordo se aplica a todas las paradas de dicha estación.

   a) *Tipo de campo:* Numérico (Llave foránea)

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* No

2. **to_stop_id (necesario):** Contiene un ID de parada que identifica una parada o estación en la que termina una conexión entre rutas. Estos también se encuentran en la tabla stops.

Si el ID de parada hace referencia a una estación que contiene varias paradas, esta regla de transbordo se aplica a todas las paradas de dicha estación.

a) *Tipo de campo:* Numérico (Llave foránea)

b) ***Utilizado como índice:*** No

c) *Puede ser blanco:* No

3. **transfer_type (necesario):** Especifica el tipo de conexión del par especificado (from_stop_id, to_stop_id). Los valores válidos para este campo son:

   I. 0 o (vacío): punto de de transbordo recomendado entre dos rutas.

   II. 1: punto de transbordo sincronizado entre dos rutas. El vehículo que sale espera al que llega, dejando tiempo suficiente para que un pasajero haga transbordo entre rutas.

   III. 2: este transbordo requiere una cantidad mínima de tiempo entre la llegada y la salida para garantizar la conexión. El tiempo necesario para el transbordo se especifica mediante min_transfer_time.

   IV. 3: no es posible realizar transbordos entre rutas en esta ubicación.

   a) *Tipo de campo:* Numérico

   b) ***Longitud máxima:*** 1

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* No

4. **min_transfer_time (opcional):** Cuando una conexión entre rutas requiere una cantidad de tiempo entre la llegada y la salida (transfer_type=2), el campo min_transfer_time define la cantidad de tiempo que debe estar disponible en un itinerario para permitir la realización de transbordos entre rutas en las paradas correspondientes. El tiempo expresado mediante min_transfer_time debe ser suficiente para permitir que un usuario normal se desplace de una parada a otra, incluido un poco de tiempo extra de tiempo establecido para las posibles variaciones de horario en cada ruta.

   El valor min_transfer_time se debe ingresar en segundos y debe ser un entero no negativo.

   a) *Tipo de campo:* Numérico

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* Sí

En la figura 4.12 se muestra el modelo UML de esta tabla junto con sus relaciones

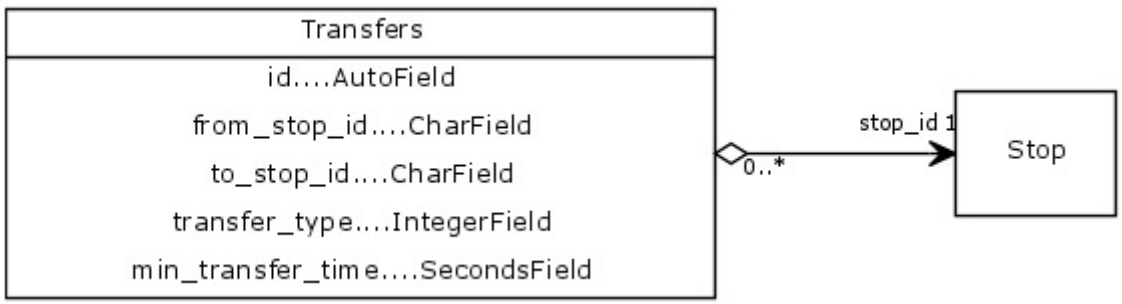

Figura 4.12: UML de la tabla Transfers de la base de datos

4.3.13. Viaje (Trip)

Define el modelo que genera el archivo trips.txt de la especificación. Este es un archivo necesario según los documentos de referencia. [19]

La base de datos maneja una tabla independiente para este archivo con los siguientes atributos en ella:

1. **route_id (necesario):** Identifica una ruta de forma exclusiva. Este valor se encuentra en la tabla routes
   a) *Tipo de campo:* Numérico (Llave foránea)
   b) ***Utilizado como índice:*** No
   c) *Puede ser blanco:* No
2. **service_id (necesario):** Identifica de forma exclusiva un conjunto de fechas en el que el servicio se encuentra disponible en una o más rutas. Este valor se encuentra en la tabla calendar o calendar_dates.
   a) *Tipo de campo:* Numérico (Llave foránea)
   b) ***Utilizado como índice:*** No
   c) *Puede ser blanco:* No
3. **trip_id (necesario):** Identifica un viaje. trip_id es un conjunto de datos único.
   a) *Tipo de campo:* Cadena
   b) ***Longitud máxima:*** 255
   c) ***Utilizado como índice:*** Sí
   d) *Puede ser blanco:* No
4. **trip_headsign (opcional):** Contiene el texto que aparece en un cartel que identifica el destino del viaje para los pasajeros. Se puede utilizar para distinguir diferentes patrones de servicio dentro de la misma ruta. Si la señal de destino cambia durante un viaje, puedes

anular el valor de trip_headsign al especificar valores para el campo stop_headsign en stop_times.

a) *Tipo de campo:* Cadena

b) ***Longitud máxima:*** 255

c) ***Utilizado como índice:*** No

d) *Puede ser blanco:* Sí

5. **trip_short_name (opcional):** Contiene el texto que aparece en horarios y carteles para que los pasajeros identifiquen el viaje, por ejemplo, para que identifiquen los números de tren en los viajes de cercanías. Si los usuarios no suelen utilizar los nombres de los viajes se puede dejar este campo en blanco.

   Un valor de trip_short_name, si se proporciona, debe identificar de forma exclusiva un viaje durante un día de servicio.

   a) *Tipo de campo:* Cadena

   b) ***Longitud máxima:*** 10

   c) ***Utilizado como índice:*** No

   d) *Puede ser blanco:* Sí

6. **direction_id (opcional):** Contiene un valor binario que indica la dirección de un viaje. Usa este campo para diferenciar viajes con dos direcciones con el mismo valor de route_id. Este campo no se debe utilizar en la elaboración de rutas, sólo proporciona una forma de diferenciar viajes por la dirección al publicar horarios. Es posible especificar nombres para cada dirección con el campo trip_headsign.

   I. 0 : viaje en una dirección (p. ej., viaje que sale).

   II. 1: viaje en la dirección opuesta (p. ej., viaje que llega).

   Por ejemplo, se pueden utilizar los campos trip_headsign y direction_id juntos para asignar un nombre a los viajes 1234 en ambas direcciones. La tabla trips contendrá las tuplas siguientes para su utilización en horarios:

   trip_id, trip_headsign, direction_id
   1234, to Airport,0
   1505, to Downtown,1

   a) *Tipo de campo:* Numérico

   b) ***Longitud máxima:*** 1

   c) ***Utilizado como índice:*** No

d) *Puede ser blanco:* Sí

7. **block_id (opcional):**

   Identifica el bloque al que pertenece el viaje. Un bloque consta de dos o más viajes secuenciales realizados en el mismo vehículo, en los que un pasajero puede cambiar de viaje si simplemente permanece en el vehículo (el caso del Metro). El valor block_id debe mencionarse en dos o más viajes de trips.

   a) *Tipo de campo:* Numérico (Llave foránea)

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* Sí

8. **shape_id (opcional):** Define la forma del viaje. Este valor se encuentra en la tabla shapes.

   a) *Tipo de campo:* Numérico (Llave foránea)

   b) ***Utilizado como índice:*** No

   c) *Puede ser blanco:* Sí

En la figura 4.13 se muestra el modelo UML de esta tabla junto con sus relaciones

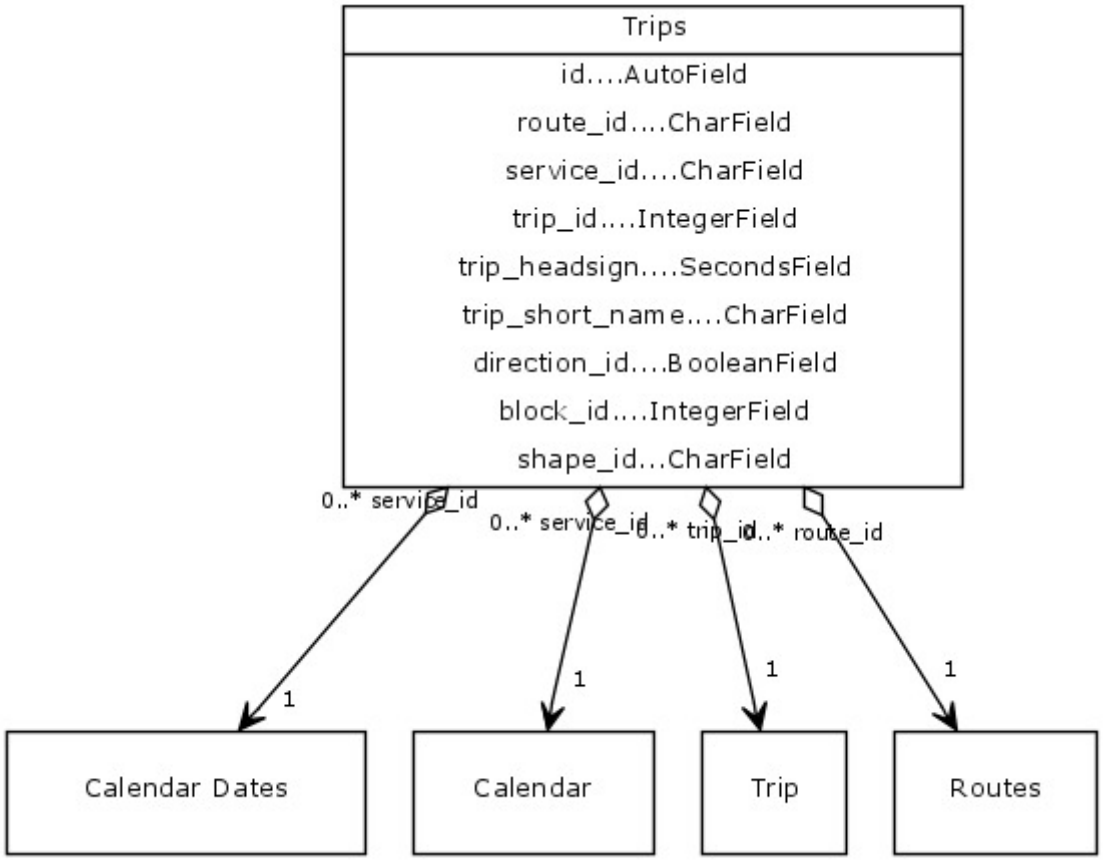

Figura 4.13: UML de la tabla Stops Times de la base de datos

4.3.14. Relaciones en GTFS

En la figura 4.14 se muestran las relaciones finales de cada una de las tablas y archivos en el estándar GTFS. En las conexiones está anotado con qué atributo están relacionados

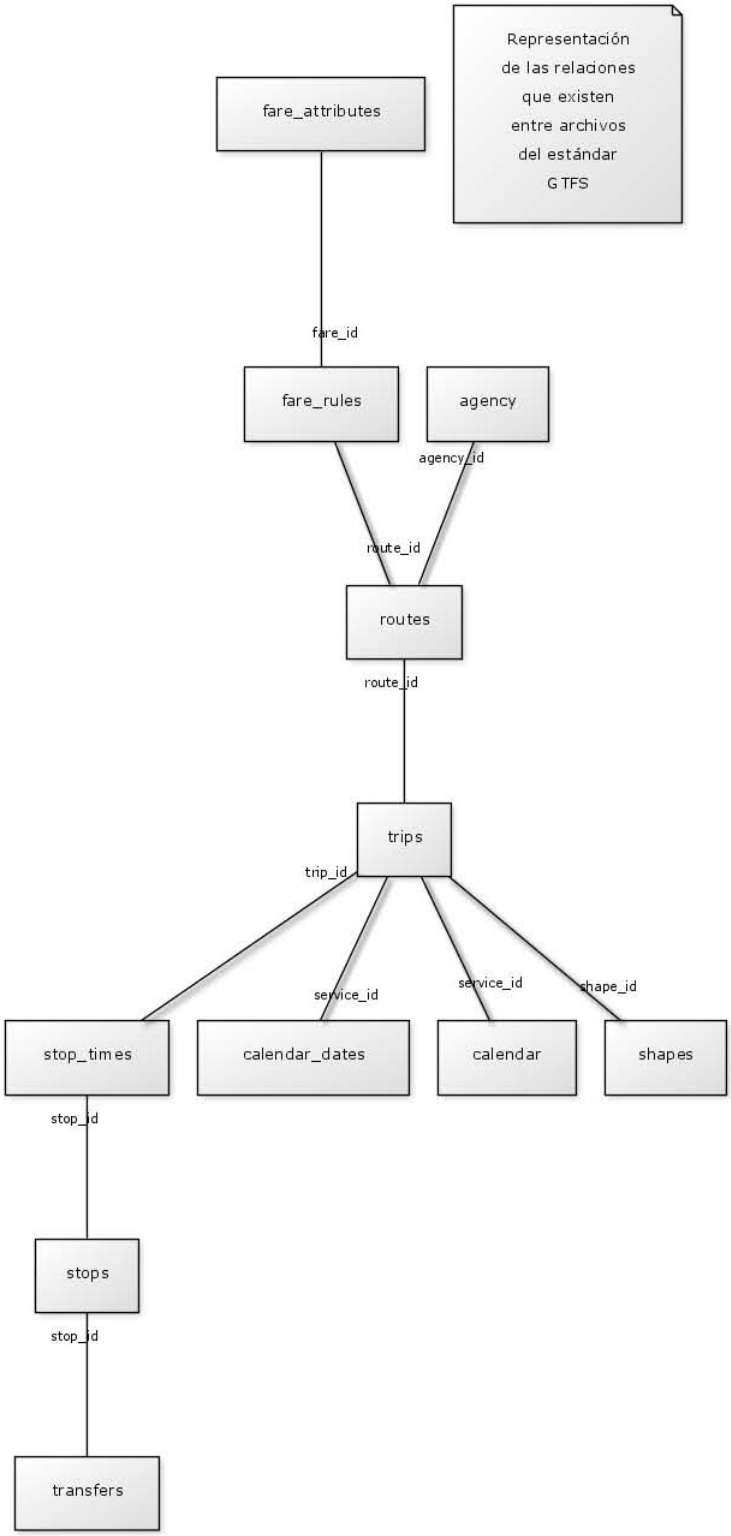

Figura 4.14: Relación existentes entre los archivos del estándar GTFS

4.4. GTFS en tiempo real

Propone un especificación para el intercambio de información geográfica en formato GTFS en tiempo real, en donde la agencia que desarrolla la información permite a los desarrolladores y el público en general acceder a la información de sus sistemas en el momento que ésta es capturada. Es diseñado bajo las mismas premisas del estándar GTFS permitiendo que ambos trabajen en conjunto. El objetivo principal de este estándar es generar consistencia entre los datos de diferentes agencias y proveedores.

Trabajar con información en tiempo real es mucho más complejo que trabajar con sistemas estáticos como los definidos en la sección anterior. Se deben de tomar en cuenta muchos más factores a la hora de publicar la información como el número de veces que cada usuario realizará peticiones por minuto para acceder a ésta.

Existen 3 diferentes tipos de información que se puede publicar y que son aceptadas por el estándar oficial

1. Actualizaciones de viajes: Permiten mandar en tiempo real reportes acerca de lo que está sucediendo dentro de una ruta, un viaje o un transporte

2. Alertas de servicios: Si sucede algo con un viaje, este feed permite mandar información a los usuarios.

3. Posición de los vehículos: Permite saber en todo momento en dónde se encuentra cada uno de los vehículos y también la cantidad de tráfico que existe en el punto en el que se encuentra.

4.4.1. Actualizaciones de viajes

Permiten mandar información de modificaciones que existen en el horario de un viaje, un ejemplo podría ser un retraso o una modificación del tiempo que normalmente se realiza entre paradas. Estas actualizaciones son pensadas para generar un horario de llegada y salida para todas las paradas en la ruta. También puede servir para establecer viajes que cambien de ruta o viajes que se cancelen.

El estándar establece que debe de haber como máximo sólo una actualización de viaje por cada viaje programado. Una actualización de viaje implica una o más actualizaciones en las paradas de los vehículos.

4.4.2. Alerta de servicios

Las alertas de servicios permite actualizar las aplicaciones con mensajes acerca de eventos importantes que suceden dentro de una ruta como interrupciones. Eventos como cancelaciones de viajes o demoras deben de ser comunicados a través de Actualizaciones de viajes.

Cada una de las alertas puede incluir diferentes campos de información:

1. La URL del sitio en el que se puede encontrar más información de la ruta
2. El texto del encabezado que resume la alerta
3. La descripción completa de la alerta que se mostrará junto al encabezado.

También existen otros campos que ayudan a categorizar la alerta entre los que se encuentran:

1. **Periodo:** Este campo establece el lapso de tiempo en el cual aparecerá la alerta. Este periodo debe cubrir todo el tiempo mientras la alerta sea vigente.

   Si no existe un periodo en los datos, entonces la alerta se muestra mientras siga existiendo dentro del feed.

2. **Selector de entidad:** Especifica exactamente qué partes de la red están afectadas por el evento que provocó la alerta, esto permite seccionar la red de transporte y enviar sólo a los usuarios que necesitan la alerta. Se pueden incluir varios selectores en caso de que se afecten múltiples entidades.

   Los identificadores que se utilizan son:

   a) ***Empresa:*** Afecta a toda la red
   b) ***Ruta:***Afecta a toda la ruta
   c) ***Tipo de ruta:*** Afecta a cualquier ruta del tipo seleccionado
   d) ***Viaje:***Afecta a un viaje en particular
   e) ***Parada:***Afecta a una parada particular

3. **Causa:** Especifica cuál es la causa de la alerta, entre las posibles opciones existen:

   a) Causa desconocida
   b) Otra causa (distinta a la presentadas)
   c) Problema técnico
   d) Huelga
   e) Manifestación
   f) Accidente
   g) Día feriado
   h) Clima
   i) Tareas de mantenimiento
   j) Tareas de construcción
   k) Actividad policial
   l) Emergencia médica

4. **Efecto:** Plantea los efectos que puede tener el problema en el sistema de transporte. Entre los que se pueden escoger se tienen:

   a) Sin servicio

   b) Servicio reducido

   c) Demoras importantes (aquellas poco importantes se deben informar a través del sistema de Actualizaciones de viaje)

   d) Desvío

   e) Servicio adicional

   f) Servicio modificado

   g) Traslado de parada

   h) Otro efecto (distinto a los presentados)

   i) Efecto desconocido

4.4.3. Posición del vehículo

Este tipo de feed está pensado para ser un medio de publicación generado automáticamente a través de un dispositivo con GPS dentro del vehículo.

El viaje que está realizando el vehículo se maneja a través del campo de descriptor de viaje, el cual se puede relacionar con la tabla de viaje dentro del sistema GTFS. Al mismo tiempo se puede proporcionar la descripción del vehículo que está enviando la información.

Un campo opcional que está permitido es el tiempo en el que se tomó la medida, aunque será diferente al tiempo que se debe enviar en el encabezado del feed, permite saber los momento en los que cada una de las mediciones.

Finalmente se puede proporcionar el campo llamado stop_id o stop_sequence que implica la parada a la que el vehículo se está dirigiendo o a la que está llegando.

4.5. Modelado del feed GTFS-RT

Los sistemas en tiempo real no requieren necesariamente de una base de datos a menos que se quiera guardar un historial de cómo se veía una sección del feed en un momento dado.

Para el modelado del sistema en tiempo real, algunas tablas fueron colapsadas para reducir el número de operaciones de unión a la hora de consultar la base de datos. Entre las tablas conjuntas se tiene:

1. **TripUpdate.trip:** Este campo maneja los atributos trip_id, trip_start_time, trip_start_date

2. **TripUpdate.vehicle:** Este campo maneja los atributos vehicle_id, vehicle_label, vehicle_license_plate

3. **StopTimeUpdate.arrival:** Maneja los atributos arrival_time, arrival_delay y arrival_ui
4. **StopTimeUpdate.departure:** Maneja el atributo departure_time.
5. **Alert.active_period:** Es reducido a sólo los campos Alert.start y Alert.end. Si existen múltiples periodos, entonces sólo el primero es guardado.
6. **Position.latitude:** Maneja el atributo position_latitude
7. **Position.longitude:** Maneja el atributo position_longitude
8. **Position.bearing:** Maneja el atributo position_bearing
9. **Position.speed:** Maneja el atributo position_speed
10. **VehicleDescriptor.id:** Maneja el atributo vehicle_id
11. **VehicleDescriptor.label:** Maneja el atributo vehicle_label
12. **VehicleDescriptor.license_plate:** Maneja el atributo vehicle_license_plate

Para mostrar la evolución en el diseño de un sistema feed para tiempo real, se diseñarán esquemas a lo largo de esta sección que muestren cómo se relacionan sus partes.

Los feed en tiempo real se dividen en dos secciones, las cuales son:

1. **header:** Son los metadatos de un feed. Contienen toda la información necesaria para poder trabajar con los datos.
2. **entity:** Representa un conjunto de datos que están agrupados bajo una identidad común. Por ejemplo, datos relacionados con el mismo viaje. Pueden haber más de una ola entidad en cada uno de los feed.

En la figura 4.15 se muestra la estructura básica que un feed en tiempo real debe de tener

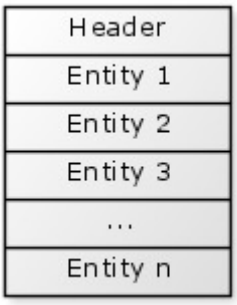

Figura 4.15: Estructura básica de un GTFS en tiempo real

4.5.1. Cabecera (Header)

En la cabecera es donde se escriben los metadatos del feed que serán utilizados por los interpretes para poder entender cada una de las entidades.

1. **gtfs_realtime_version (necesario):** Especifica la versión del feed. La versión actual es 1.0

   a) *Tipo de campo:* Cadena

   b) *Puede ser blanco:* No

2. **incrementality (opcional):** Determina si la información es incremental. Puede llegar a tener dos valores los cuales pueden ser:

   a) ***FULL_DATASET:*** La actualización del feed sobrescribirá toda la información en tiempo real anterior. Por lo tanto, se espera que ésta actualización proporcione un resumen completo de toda la información en tiempo real conocida.

   b) ***DIFFERENTIAL:*** En este momento, este modo no está admitido y su comportamiento no se especifica para los feeds que usan este modo.

   a) *Tipo de campo:* Cadena

   b) *Puede ser blanco:* Sí

3. **timestamp (opcional):** Este campo es utilizado para identificar el tiempo en que se ha creado el contenido del feed.

   a) *Tipo de campo:* Uint64

   b) *Puede ser blanco:* Sí

En la figura 4.16 se muestra la estructura básica de un feed GTFS con la cabecera incluida

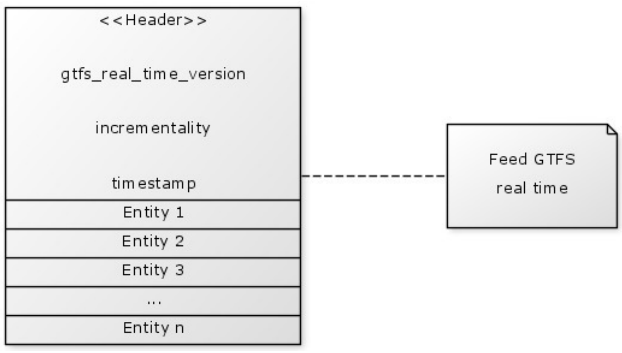

Figura 4.16: Estructura básica de un GTFS en tiempo real

4.5.2. Entidad (Entity)

Una entidad puede ser descrita como un conjunto de datos con características comunes que contienen los tres tipos de actualizaciones que se pueden enviar trip_update, vehicle y alert.

1. **id (necesario):** Identifica de manera única una entidad en el feed. Se usan sólo para proporcionar soporte de incrementos, ya que ayudan a saber qué feeds fueron ya descargados. Las entidades reales como vehículos y viajes se hacen referencia a través de otros campos.
   a) *Tipo de campo:* Cadena
   b) *Puede ser blanco:* No
2. **is_deleted (opcional):** Sirve para saber si la entidad debe eliminarse. Es importante cuando se hacen búsquedas incrementales.
   a) *Tipo de campo:* Boolean
   b) *Puede ser blanco:* Sí
3. **trip_update (opcional):** Datos sobre demoras de salida en tiempo real de un viaje.
   a) *Tipo de campo:* TripUpdate (Referencia a otro objeto)
   b) *Puede ser blanco:* Sí
4. **vehicle (opcional):** Datos sobre la posición en tiempo real de un viaje.
   a) *Tipo de campo:* VehiclePosition (Referencia a otro objeto)
   b) *Puede ser blanco:* Sí
5. **alert (opcional):** Datos sobre alertas en tiempo real
   a) *Tipo de campo:* Alert (Referencia a otro objeto)
   b) *Puede ser blanco:* Sí

En la figura 4.17 se puede ver un esquema más completo de cómo se ve el feed GTFS

4.5.3. Actualizaciones de Viajes (Trip Update)

Feed que controla la actualización en tiempo real del progreso del transporte en un viaje. Según el valor del campo ScheduleRelationship que se encuentra dentro del objeto StopTimeUpdate, TripUpdate puede significar lo siguiente:

I. Un viaje que avanza según la programación.

II. Un viaje que avanza por una ruta, pero que no tiene una programación fija.

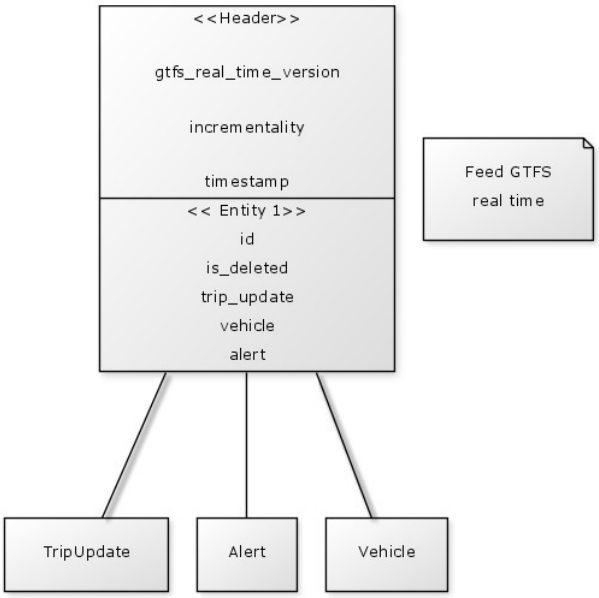

Figura 4.17: Estructura de cabecera y entidad de un GTFS real time

III. Un viaje que se ha agregado o se ha quitado en relación con una programación.

Las actualizaciones pueden ser para eventos de llegada o salida futuros y previstos, o para eventos pasados que ya ocurrieron. En la mayoría de los casos, la información sobre los eventos pasados es un valor medido, por lo tanto, se recomienda que su valor de incertidumbre sea 0. Aunque puede haber algunos casos en que esto no sea así, por lo que se admiten valores de incertidumbre distintos de 0 para los eventos pasados. Si el valor de incertidumbre de una actualización no es 0, entonces la actualización es una predicción aproximada para un viaje que no se ha completado o la medición no es precisa o la actualización fue una predicción para el pasado que no se ha verificado después de que ocurrió el evento.

Se debe tener en cuenta que la actualización puede describir un viaje que ya se ha completado. En este caso, es suficiente con proporcionar una actualización para la última parada del viaje.

1. **trip (necesario):** El viaje al cual se aplica este mensaje. Puede haber una entidad de TripUpdate, como máximo, por cada instancia de viaje real. Si no hay ninguna, entonces no habrá información de predicciones disponible.

   a) *Tipo de campo:* TripDescriptor (Referencia a otro objeto)

   b) *Puede ser blanco:* No

2. **vehicle (opcional):** Información adicional sobre el vehículo con el cual se está realizando el viaje.

   a) *Tipo de campo:* VehicleDescriptor (Referencia a otro objeto)

   b) *Puede ser blanco:* Sí

3. **stop_time_update (opcional):** Las actualizaciones de StopTimes para el viaje. Las actualizaciones deben de ordenarse por secuencia de parada y deben de aplicarse a todas las siguientes paradas del viaje hasta la próxima especificada.

   a) *Tipo de campo:* StopTimeUpdate (Referencia a otro objeto)

   b) *Puede ser blanco:* Sí

4. **timestamp (opcional):** Representa el momento en el que se generó la actualización o en la que se descubrió la necesidad de la actualización.

   a) *Tipo de campo:* Uint64

   b) *Puede ser blanco:* Sí

En la figura 4.18 se muestra cómo se encuentra la estructura del GTFS descrito hasta ahora

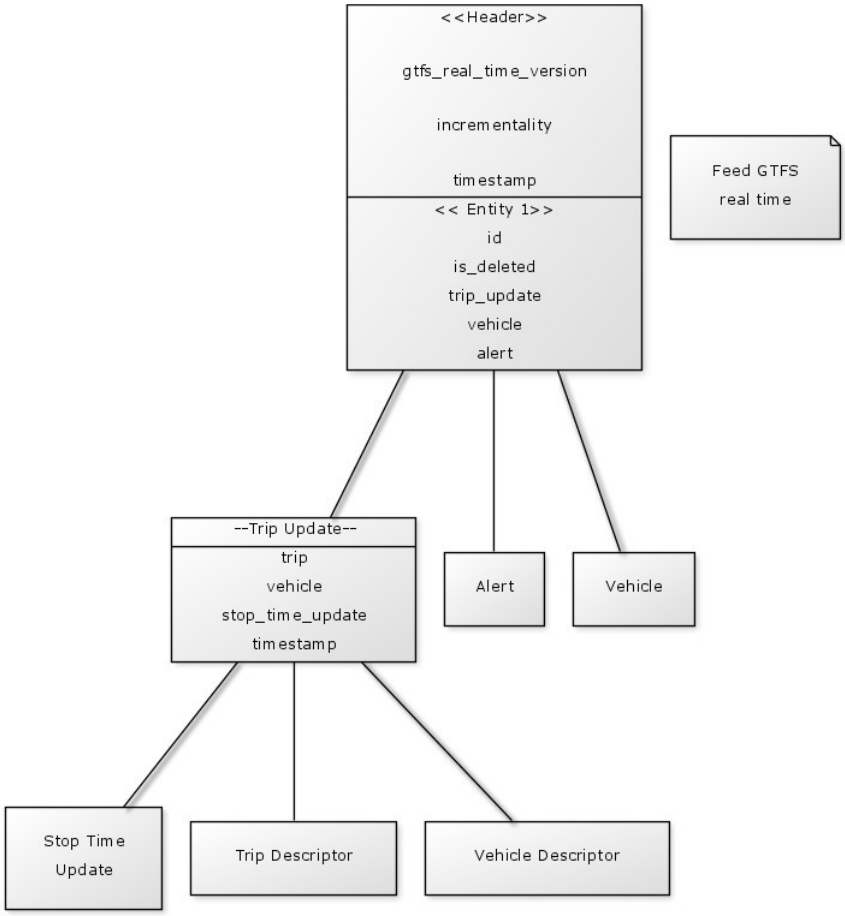

Figura 4.18: Estructura del GTFS incluyendo Trip Update

4.5.4. Actualizaciones de paradas (Stop Time Update)

Actualiza en tiempo real los eventos de llegada o de salida para un determinado viaje.

Las actualizaciones se pueden proporcionar tanto para eventos pasados como futuros.

La actualización está vinculada a una parada específica sea a través de stop_sequence o de stop_id, de manera que uno de estos campos debe definirse necesariamente.

1. **stop_sequence (opcional):** Hace referencia a la tabla stop_times del estándar GTFS explicado en la sección anterior

   a) *Tipo de campo:* uint32 (Llave foránea)

   b) *Puede ser blanco:* Sí

2. **stop_id (opcional):** Hace referencia a la tabla stops del estándar GTFS

   a) *Tipo de campo:* Cadena (Llave foránea)

   b) *Puede ser blanco:* Sí

3. **arrival (opcional):** Representa el evento de arribo a una parada.

   a) *Tipo de campo:* StopTimeEvent (Referencia a otro objeto)

   b) *Puede ser blanco:* Sí

4. **departure (opcional):** Es la actualización en el tiempo de salida de algún transporte.

   a) *Tipo de campo:* StopTimeEvent (Referencia a otro objeto)

   b) *Puede ser blanco:* Sí

5. **schedule_relationship (opcional):** Representa la relación que existe entre el evento y el vehículo. Los valores que puede tomar son:

   I. ***SCHEDULED:*** Implica que el vehículo está avanzando según su programación, aunque no necesariamente de acuerdo con los tiempos de la programación. Al menos debe de proporcionarse uno de los valores de llegada y salida.

   II. ***SKIPPED:*** La parada se omite, es decir, el transporte no se detendrá en esa parada. Los valores de llegada y salida para este campo son opcionales.

   III. ***NO_DATA:*** No se proporcionan los datos para esta parada. Lo que implica que no existe información en tiempo real. Cuando se especifica este campo en una parada, todas las demás paradas heredan este campo por lo que se puede utilizar para mencionar a partir de qué punto no se tiene información acerca de las llegadas.

   a) *Tipo de campo:* Cadena

   b) *Puede ser blanco:* Sí

En la figura 4.19 se muestra la estructura del sistema GTFS hasta el momento

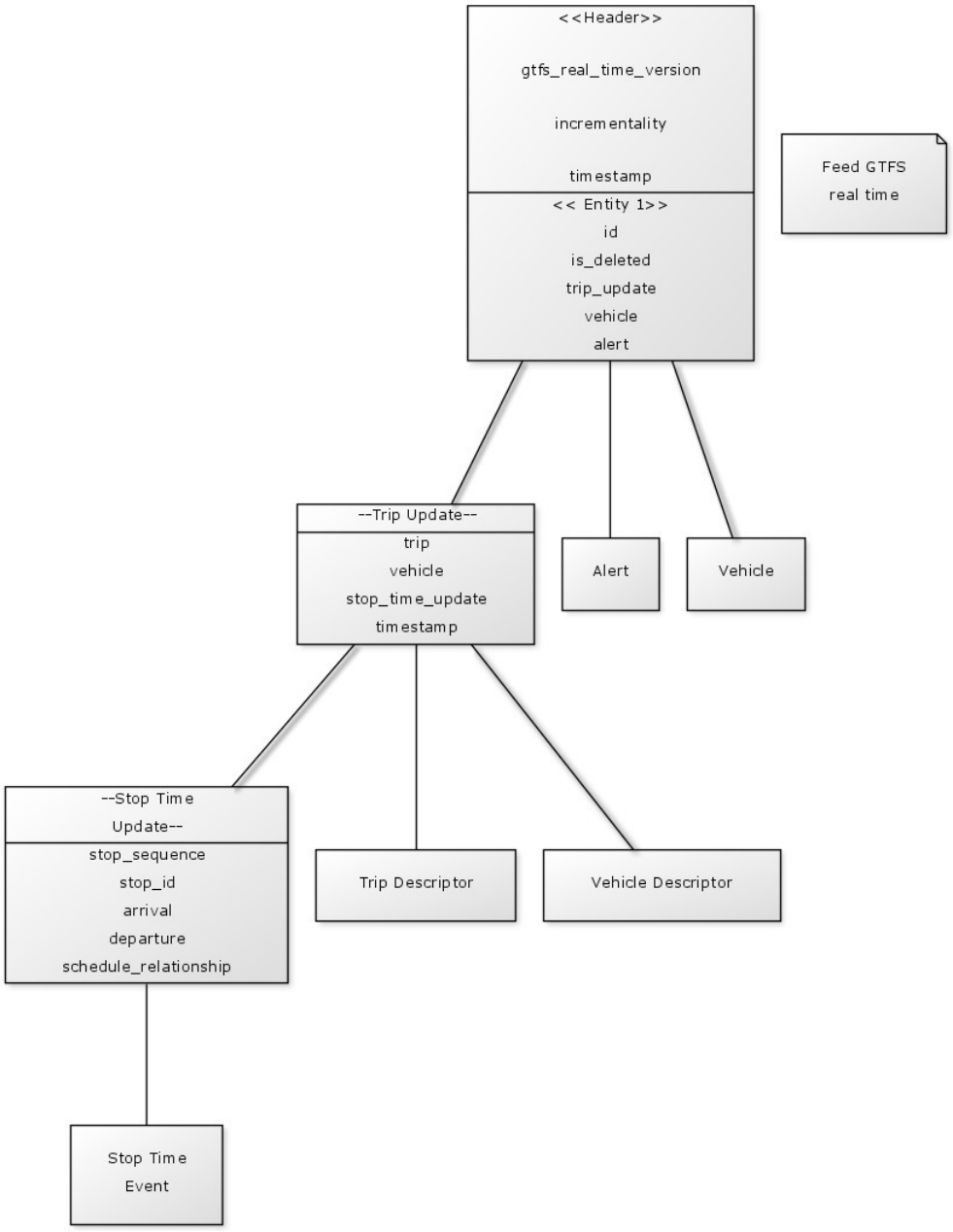

Figura 4.19: Estructura del GTFS incluyendo Stop Time Update

4.5.5. Información de horarios(Stop Time Event)

Los horarios consisten en la información sobre demoras o tiempos estimados y la incertidumbre.

I. La demora (delay) debe usarse cuando la predicción se realiza con relación a una programación existente en GTFS.

II. El tiempo (time) debe darse aunque no haya una programación prevista. Si se especifican tanto el tiempo como la demora, el tiempo será prioritario (aunque por lo general el tiempo, si se otorga para un viaje programado, debe ser igual al tiempo programado en GTFS + la demora).

La incertidumbre se aplica de la misma forma tanto al tiempo como a la demora. La incertidumbre especifica el error esperado en una demora real. Es posible que la incertidumbre sea 0, por ejemplo, para los trenes que funcionan con un control de horarios por computadora.

1. **delay (opcional):** La demora (en segundos) puede ser positiva(el vehículo está atrasado) o negativa (el transporte está adelantado). Una demora de 0 significa que el vehículo está a tiempo.

   a) *Tipo de campo:* int32

   b) *Puede ser blanco:* Sí

2. **time (opcional):** Evento como tiempo absoluto. Este es el momento en el que se prevé que suceda.

   a) *Tipo de campo:* int64

   b) *Puede ser blanco:* Sí

3. **uncertainity (opcional):** Si se omite la incertidumbre, se toma como desconocida. Si la predicción es precisa, entonces se puede dar una incertidumbre de 0.

   a) *Tipo de campo:* int32

   b) *Puede ser blanco:* Sí

El feed mostrado en la figura 4.20 muestra las conexiones actuales, junto con la relación que existe hasta el momento con el estándar GTFS (en azul)

4.5.6. Descriptor de Viajes (Trip Descriptor)

Identifica un viaje a través de la tabla trips del estándar GTFS o todas las instancias de viajes por una ruta dada. Si sólo representa un viaje, entonces el campo trip_id debe ser enviado. Si al mismo tiempo se identifica una ruta, en esta debe de encontrarse el viaje al que apunta trip_id. Para identificar todos los viajes de una ruta, entonces sólo se debe de dar route_id. Se deben de proporcionar horas absolutas de llegada o de salida.

1. **trip_id (opcional):** Hace referencia a la tabla trips del estándar GTFS. Para los viajes sin frecuencia extendida, este campo es suficiente. Para viajes con frecuencias, los campos strat_time y start_date también deben ser dados.

   a) *Tipo de campo:* Cadena (Llave foránea)

   b) *Puede ser blanco:* Sí

2. **route_id (opcional):** Hace referencia a la tabla routes del estándar GTFS

   a) *Tipo de campo:* Cadena (Llave foránea)

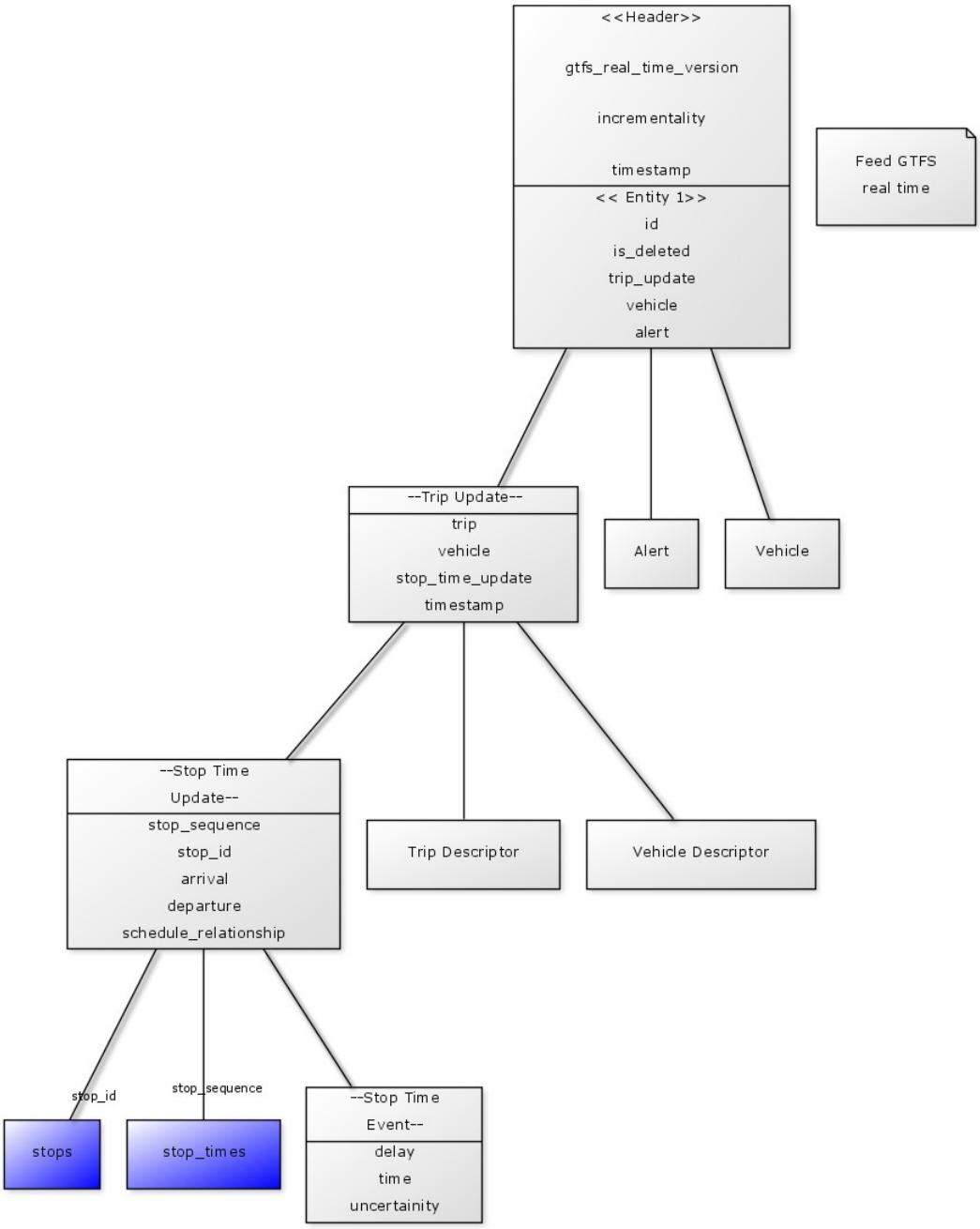

Figura 4.20: Estructura del GTFS incluyendo Stop Time Event

b) *Puede ser blanco:* Sí

3. **start_time (opcional):** Representa el momento en el que comienza el o los viajes que representa el feed. Este campo debe de ser dado sí y sólo sí el viaje es de frecuencia extendida en el feed GTFS. El formato del campo es el mismo que el de start_time de la tabla frequencies.

   a) *Tipo de campo:* Fecha

   b) *Puede ser blanco:* Sí

4. **start_date (opcional):** La fecha de inicio programada de esta instancia de viaje. Este

campo debe proporcionarse para eliminar la ambigüedad de los viajes que están tan retrasados que pueden superponerse con un viaje para el día siguiente.

a) *Tipo de campo:* Fecha

b) *Puede ser blanco:* Sí

5. **schedule_relationship (opcional):** Representa la relación entre el viaje y la programación. Si un viaje se realiza de acuerdo con la programación temporal no se refleja en el estándar GTFS. Los valores que puede tener son:

   I. ***SCHEDULED:*** Viaje que se está realizando de acuerdo con la programación establecida en las tablas de GTFS.

   II. ***ADDED:*** Un viaje adicional que se agregó además de una programación existente, por ejemplo, para reemplazar un vehículo averiado o para responder a una carga repentina de pasajeros.

   III. ***UNSCHEDULED:*** Un viaje que se está realizando sin ninguna programación asociada.

   IV. ***CANCELED:*** Una viaje que existió en la programación, pero que luego se eliminó.

   V. ***REPLACEMENT:*** Un viaje que reemplaza una parte de la programación estática. Si el selector identifica una ruta, entonces todos los viajes de la ruta se reemplazan.

   El reemplazo se aplica solamente a la parte del viaje que se suministra. Por ejemplo, se se considera una ruta que pasa por las paradas A,B,C,D,E,F y un viaje REPLACEMENT proporciona datos para las paradas A,B,C. Entonces, los horarios para las paradas D,E,F todavía se toman de la programación estática.

   a) *Tipo de campo:* Cadena

   b) *Puede ser blanco:* Sí

En la figura 4.21 se muestra la estructura del feed hasta el momento. Algunos campos fueron simplificados para un mejor entendimiento. En azul se muestran los campos del estándar GTFS que se relacionan con este subsistema.

4.5.7. Descriptor de Vehículo (Vehicle Descriptor)

Esta información permite identificar el vehículo en particular que está realizando el viaje.

1. **id (opcional):** Identificador único del vehículo. Es utilizado para dar un seguimiento del vehículo en la medida en que avanza el sistema. Para este prototipo, esta información se encuentra en la tabla vehicles.

   a) *Tipo de campo:* Cadena (Llave foránea)

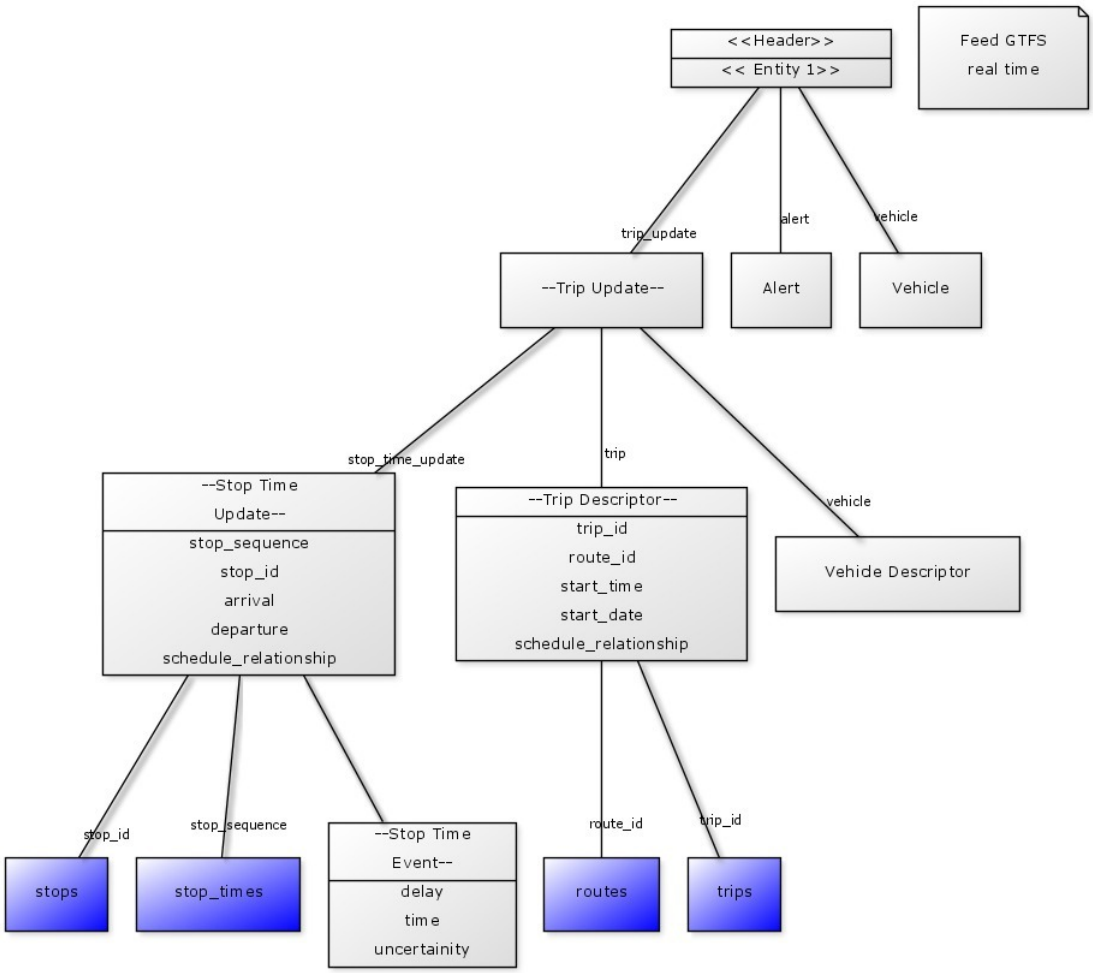

Figura 4.21: Estructura del GTFS incluyendo Trip Descriptor

b) *Puede ser blanco:* Sí

2. **label (opcional):** Etiqueta que el usuario ve en el sistema ayudando a identificar el vehículo correcto.

   a) *Tipo de campo:* Cadena

   b) *Puede ser blanco:* Sí

3. **license_plate (opcional):** La placa del vehículo.

   a) *Tipo de campo:* Cadena

   b) *Puede ser blanco:* Sí

En la figura 4.22 se muestra la estructura del feed hasta el momento.

4.5.8. Vehículos (Vehicle)

Incluye la información de la posición actual del vehículo.

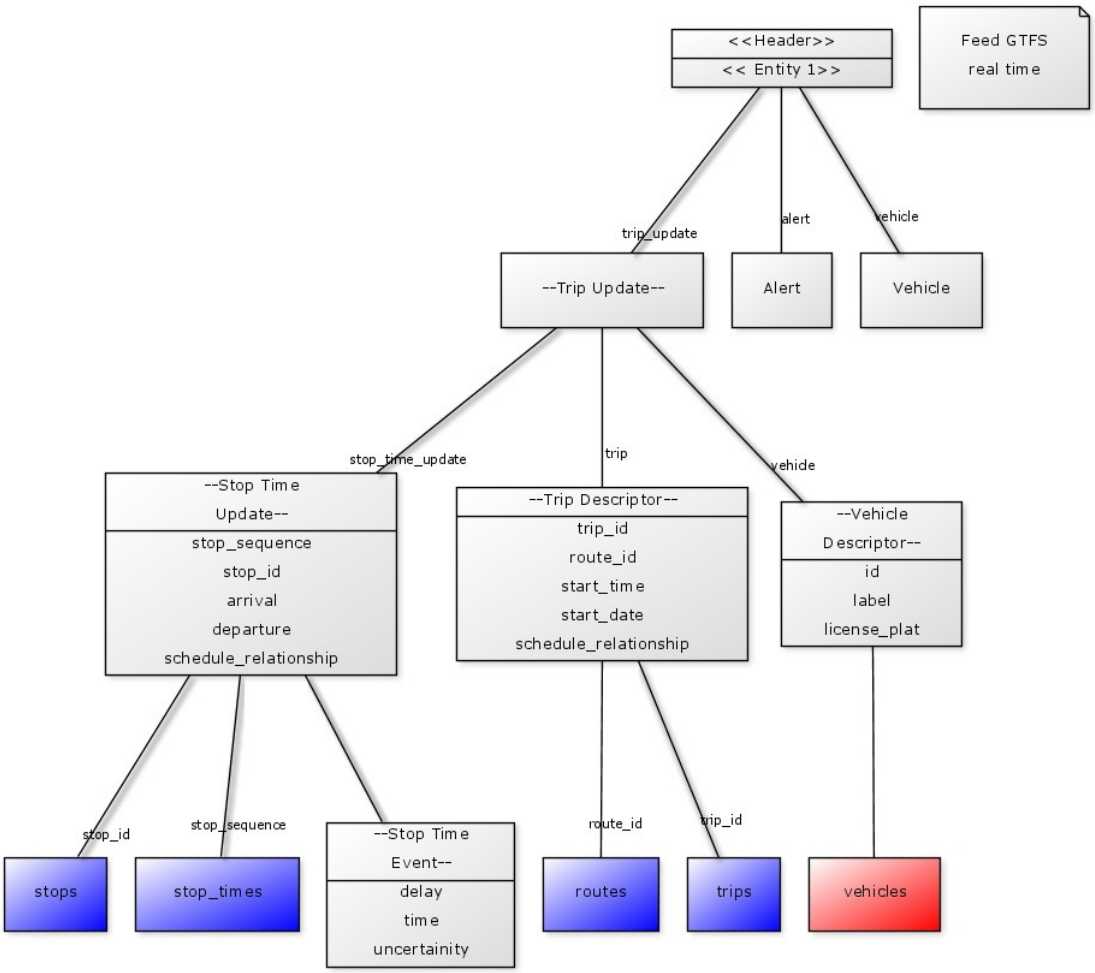

Figura 4.22: Estructura del GTFS incluyendo Vehicle Descriptor

1. **trip (opcional):** Hace referencia a la tabla trips del estándar GTFS. Representa el viaje que está haciendo el vehículo. Puede estar vacío si el vehículo no puede puede identificarse con un viaje.

   a) *Tipo de campo:* Cadena (Llave foránea)

   b) *Puede ser blanco:* Sí

2. **vehicle (opcional):** Información adicional sobre el vehículo que está realizando el viaje.

   a) *Tipo de campo:* VehicleDescriptor (Referencia a otro objeto)

   b) *Puede ser blanco:* Sí

3. **position (opcional):** Posición actual del vehículo.

   a) *Tipo de campo:* Position (Referencia a otro objeto)

   b) *Puede ser blanco:* Sí

4. **current_stop_sequence (opcional):** Representa el índice de la secuencia de parada en la que se encuentra el vehículo. El significado de este campo está determinado por el

valor de current_status.

   a) *Tipo de campo:* Position (Referencia a otro objeto)
   b) *Puede ser blanco:* Sí

5. **stop_id (opcional):** Hace referencia a la tabla stops. Identifica la parada actual.

   a) *Tipo de campo:* Cadena (Llave Foránea)
   b) *Puede ser blanco:* Sí

6. **current_status (opcional):** El estado exacto del vehículo con respecto a la parada actual. Se ignora si falta el valor en current_stop_sequence. Los valores que puede obtener son:

   I. ***INCOMING_AT:*** El vehículo está apunto de llegar a la parada.
   II. ***STOPPPED_AT:*** El vehículo está detenido en la parada.
   III. ***IN_TRANSIT_TO:*** El vehículo ha salido de la parada anterior y está en tránsito (default).

   a) *Tipo de campo:* Cadena
   b) *Puede ser blanco:* Sí

7. **timestamp (opcional):** Momento en el cual el sistema de rastreo midió su posición. Debe ser puesto en tiempo POSIX.

   a) *Tipo de campo:* Fecha
   b) *Puede ser blanco:* Sí

8. **congestion_level (opcional):** Sirve para poder indicar cuánto tráfico existe en la ruta en la que se encuentra el vehículo. Los posibles valores son:

   I. UNKNOWN_CONGESTION_LEVEL
   II. RUNNING_SMOOTHLY
   III. STOP_AND_GO
   IV. CONGESTION
   V. SEVERE_CONGESTION

   En donde SEVERE_CONGESTION representa un tráfico tal, que los automovilistas están saliendo de sus vehículos.

   a) *Tipo de campo:* Cadena
   b) *Puede ser blanco:* Sí

En la figura 4.23 se muestra la estructura del feed hasta el momento.

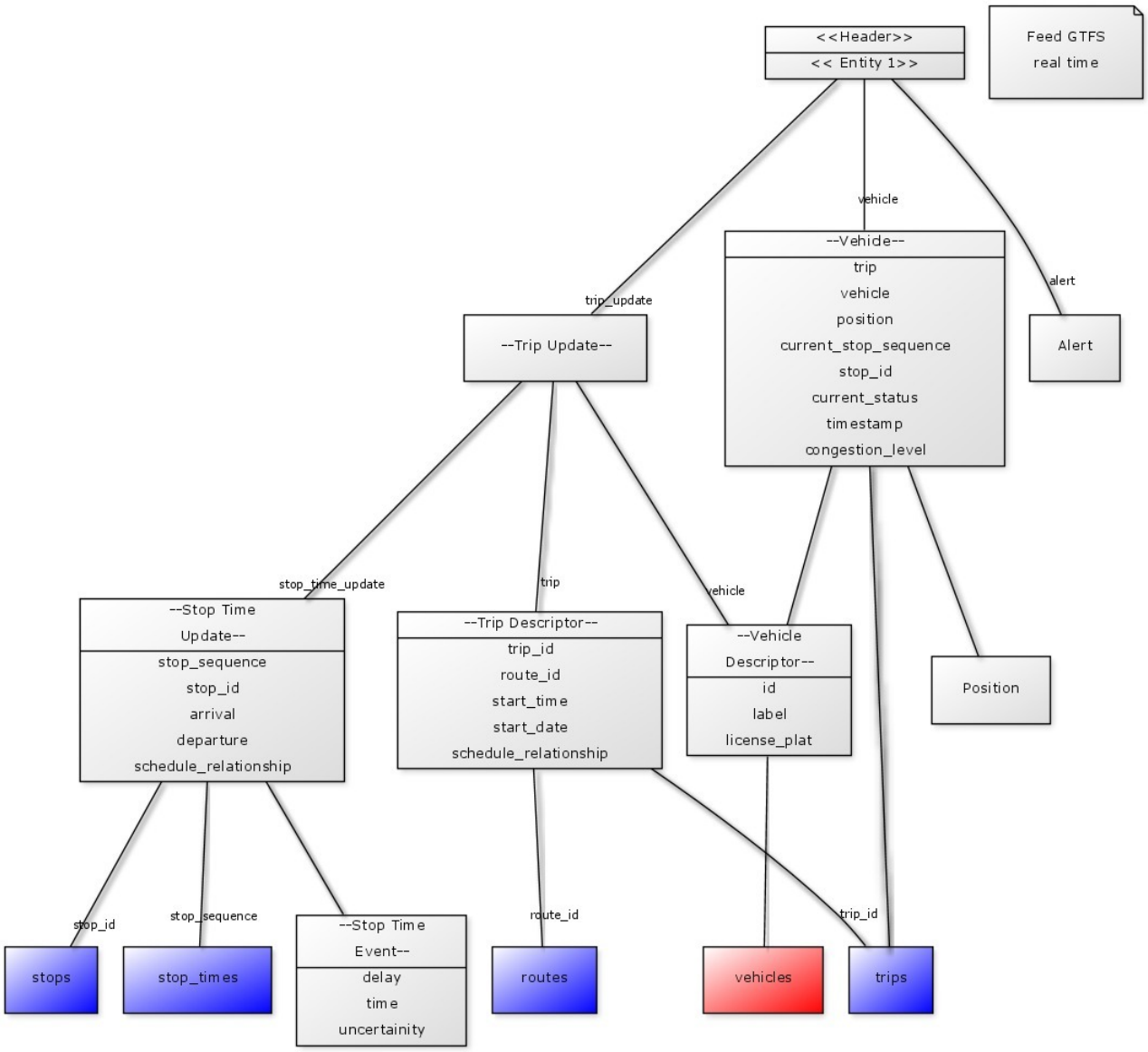

Figura 4.23: Estructura del GTFS incluyendo Vehicle

4.5.9. Posición (Position)

Contiene la posición geográfica del vehículo.

1. **latitude (necesario):** Son los grados hacia el norte en coordenadas WGS-84 [41]
   a) *Tipo de campo:* Numérico
   b) *Puede ser blanco:* No
2. **longitude (necesario):** Son los grados hacia el este en coordenadas WGS-84 [41]
   a) *Tipo de campo:* Numérico
   b) *Puede ser blanco:* No
3. **bearing (opcional):** Orientación en grados en el sentido de las agujas del reloj desde el norte verdadero, es decir 0 es el norte y 90 es el este. Esta puede ser la orientación de la

brújula o la dirección hacia la próxima parada o la ubicación intermedia.

   a) *Tipo de campo:* Numérico

   b) *Puede ser blanco:* Sí

4. **odometer (opcional):** El valor del odómetro en metros.

   a) *Tipo de campo:* Numérico

   b) *Puede ser blanco:* Sí

5. **speed (opcional):** Velocidad instantánea medida en el vehículo en metros por segundo.

   a) *Tipo de campo:* Numérico

   b) *Puede ser blanco:* Sí

En la figura 4.24 se muestra la estructura del feed hasta el momento.

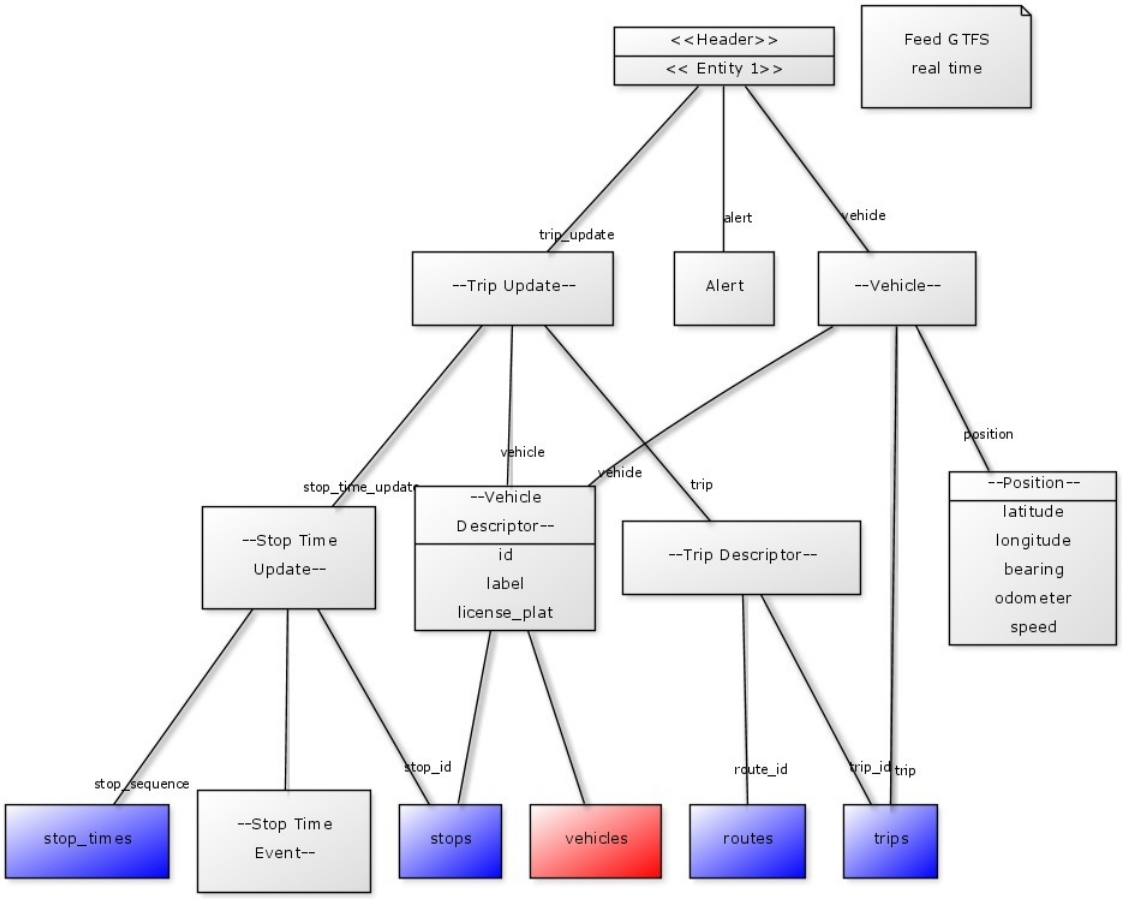

Figura 4.24: Estructura del GTFS incluyendo Position

4.5.10. Alertas (Alert)

Este feed es utilizado para el envío de alertas e incidentes dentro de la red de transporte.

1. **active_period (opcional):** Tiempo durante el cual debe mostrarse la alerta al usuario. Si falta, la alerta se mostrará durante todo el tiempo que aparezca en el feed.
   a) *Tipo de campo:* TimeRange (Referencia a otro objeto)
   b) *Puede ser blanco:* Sí
2. **informed_entity (opcional):** Entidades a cuyos usuarios debemos notificar esta alerta.
   a) *Tipo de campo:* EntitySelector (Referencia a otro objeto)
   b) *Puede ser blanco:* Sí
3. **cause (opcional):** Representa la causa que generó la alerta. Entre los valores que puede tener están:
   I. UNKNOWN_CAUSE
   II. OTHER_CAUSE
   III. TECHNICAL_PROBLEM
   IV. STRIKE
   V. DEMONSTRATION
   VI. ACCIDENT
   VII. HOLIDAY
   VIII. WEATHER
   IX. MAINTENANCE
   X. CONSTRUCTION
   XI. POLICE_ACTIVITY
   XII. MEDICAL_EMERGENCY
4. **effect (opcional):** El efecto del problema que genera la alerta. Los valores que puede tener son:
   I. NO_SERVICE
   II. REDUCED_SERVICE
   III. SIGNIFICANT_DELAYS
   IV. DETOUR
   V. ADDITIONAL_SERVICE
   VI. MODIFIED_SERVICE
   VII. OTHER_EFFECT

VIII. UNKNOWN_EFFECT

IX. STOP_MOVED

5. **url (opcional):** La URL en donde se puede encontrar información adicional sobre la alerta.

   a) *Tipo de campo:* Cadena

   b) *Puede ser blanco:* Sí

6. **header_text (opcional):** Encabezado de la alerta. Esta es el título que se mostrará resaltado.

   a) *Tipo de campo:* Cadena

   b) *Puede ser blanco:* Sí

7. **description_text (opcional):** Descripción de la alerta. La información de la descripción sirve para dar más información acerca de la alerta al usuario.

   a) *Tipo de campo:* Cadena

   b) *Puede ser blanco:* Sí

En la figura 4.25 se muestra la estructura del feed hasta el momento.

4.5.11. Selector de entidad (Entity Selector)

Campo utilizado para seleccionar una entidad en un feed GTFS. Los valores de los campos deben coincidir con los campos de la base de datos del estándar GTFS.

1. **agency_id (opcional):** Hace referencia a la tabla agency del estándar GTFS

   a) *Tipo de campo:* Cadena (Llave foránea)

   b) *Puede ser blanco:* Sí

2. **route_id (opcional):** Hace referencia a la tabla routes del estándar GTFS

   a) *Tipo de campo:* Cadena (Llave foránea)

   b) *Puede ser blanco:* Sí

3. **route_type (opcional):** Hace referencia a la tabla routes del estándar GTFS

   a) *Tipo de campo:* int32 (Llave foránea)

   b) *Puede ser blanco:* Sí

4. **trip (opcional):** Descriptor del viaje que representa el feed

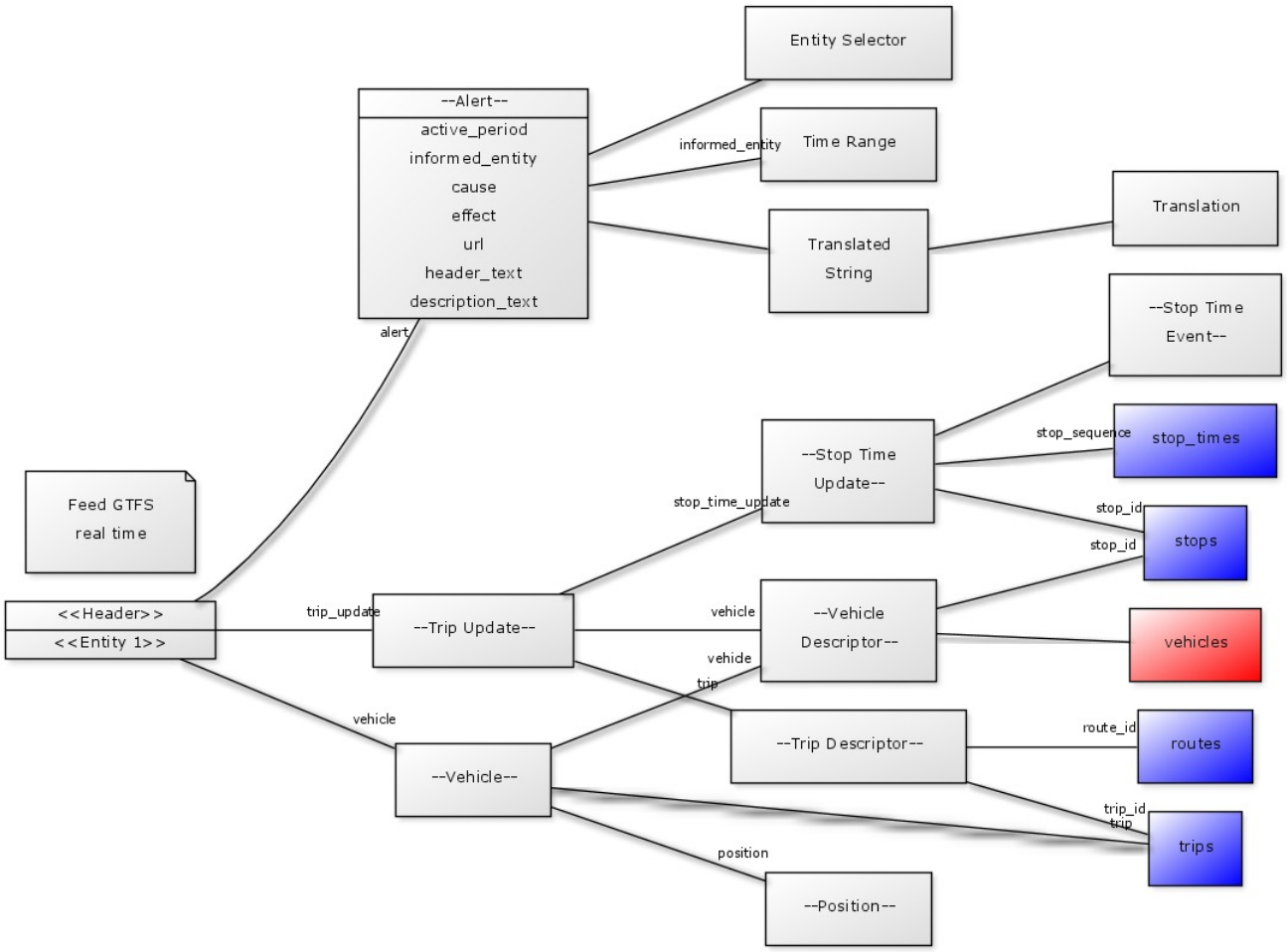

Figura 4.25: Estructura del GTFS incluyendo Alert

a) *Tipo de campo:* TripDescriptor (Referencia a otro objeto)

b) *Puede ser blanco:* Sí

5. **stop_id (opcional):** Hace referencia a la tabla stops del estándar GTFS

   a) *Tipo de campo:* Cadena (Llave foránea)

   b) *Puede ser blanco:* Sí

En la figura 4.26 se muestra la estructura del feed hasta el momento.

4.5.12. Intervalo de tiempo (Time Range)

Sirve para manejar intervalos de tiempo. Se consideran activos si el intervalo es mayor a la hora de inicio y menor a la hora final

1. **start (opcional):** Hora de inicio, debe ser representada en tiempo POSIX. Si falta esta información, el intervalo comienza en infinito negativo.

2. **end (opcional):** Hora de finalización, deber ser representada en tiempo POSIX. Si falta esta información, el intervalo finaliza con el valor infinito positivo.

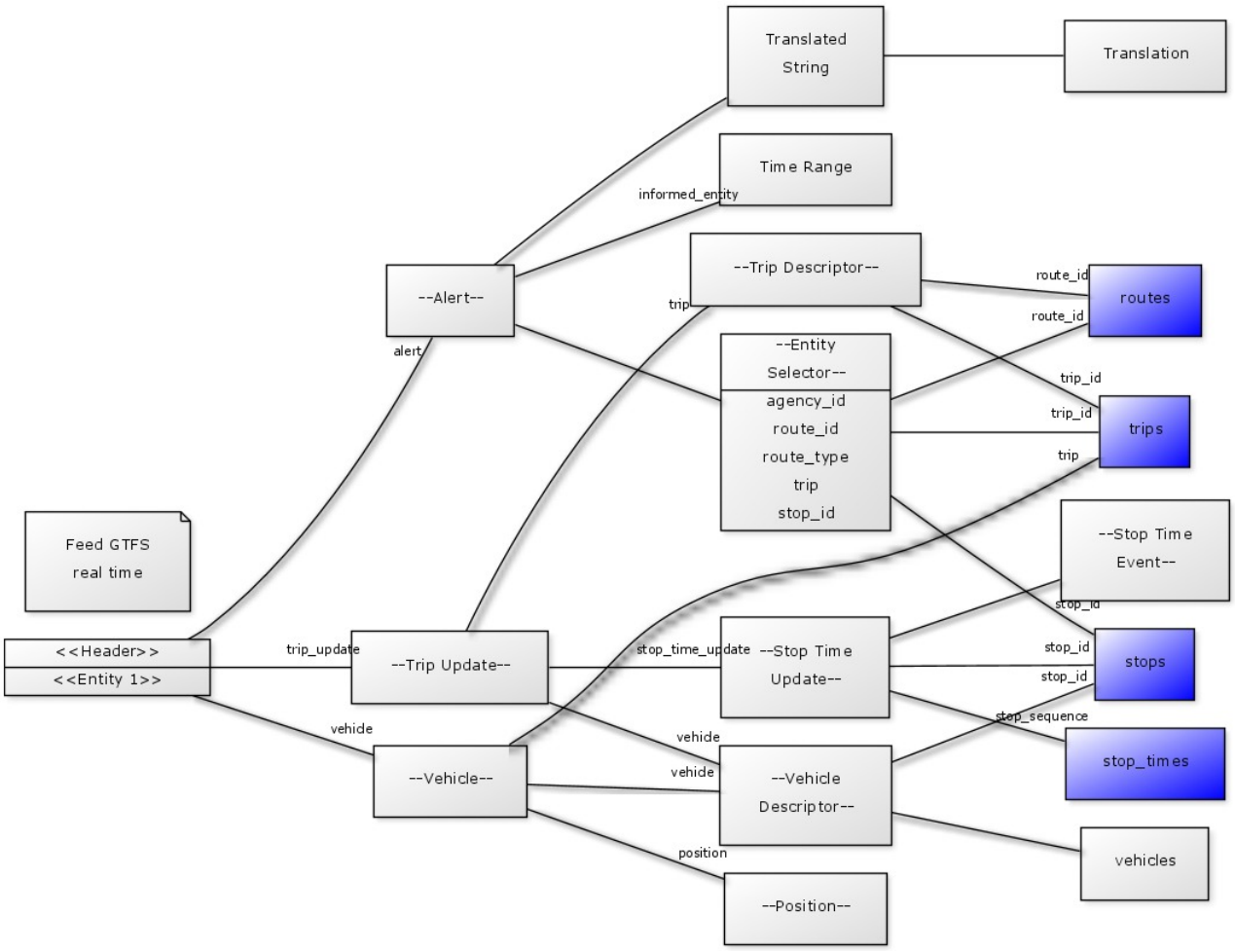

Figura 4.26: Estructura del GTFS incluyendo Entity Selector

En la figura 4.27 se muestra la estructura final del feed GTFS en tiempo real junto con el feed GTFS. En azul se muestran las tablas de la base de datos que son utilizadas para crear los archivos del estándar GTFS y en gris los archivos del GTFS en tiempo real que son creados de la base de datos.

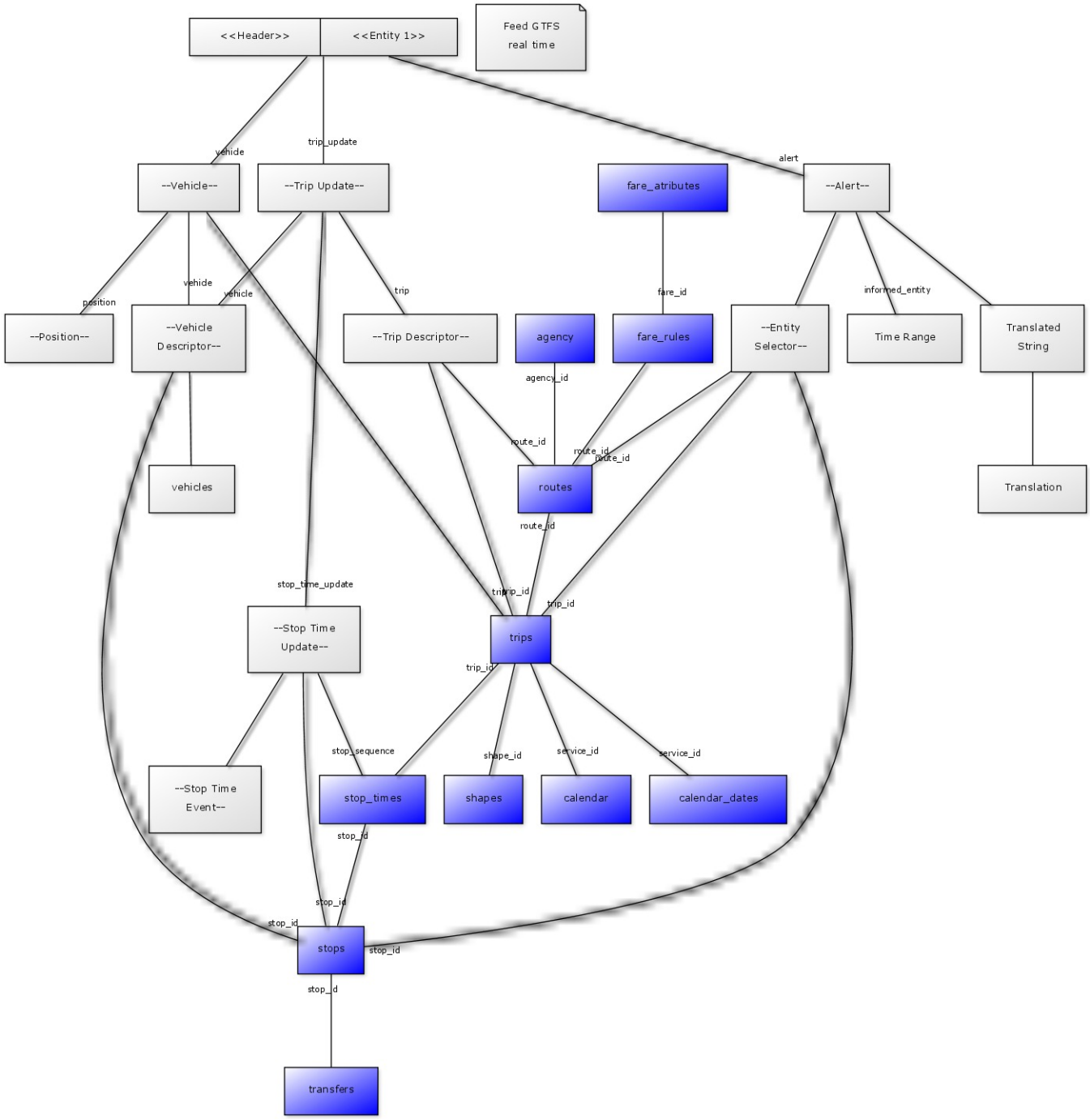

Figura 4.27: Estructura del GTFS en tiempo real final

4.6. Vistas de la base de datos

En las figuras siguientes se muestran algunas vistas de la base de datos funcionando utilizando el GTFS del gobierno de la Ciudad de México. En algunas de las tablas se muestran sólo algunas columnas por necesidad de espacio debido a que los campos son demasiado grandes y no se visualizaban correctamente.

```
                    List of relations
 Schema |                Name                 |   Type   |  Owner
--------+-------------------------------------+----------+----------
 public | agency                              | table    | otero
 public | agency_id_seq                       | sequence | otero
 public | alerts                              | table    | otero
 public | alerts_oid_seq                      | sequence | otero
 public | auth_group                          | table    | otero
 public | auth_group_id_seq                   | sequence | otero
 public | auth_group_permissions              | table    | otero
 public | auth_group_permissions_id_seq       | sequence | otero
 public | auth_permission                     | table    | otero
 public | auth_permission_id_seq              | sequence | otero
 public | auth_user                           | table    | otero
 public | auth_user_groups                    | table    | otero
 public | auth_user_groups_id_seq             | sequence | otero
 public | auth_user_id_seq                    | sequence | otero
 public | auth_user_user_permissions          | table    | otero
 public | auth_user_user_permissions_id_seq   | sequence | otero
 public | calendar                            | table    | otero
 public | calendar_dates                      | table    | otero
 public | django_admin_log                    | table    | otero
 public | django_admin_log_id_seq             | sequence | otero
 public | django_content_type                 | table    | otero
 public | django_content_type_id_seq          | sequence | otero
 public | django_session                      | table    | otero
 public | django_site                         | table    | otero
 public | django_site_id_seq                  | sequence | otero
 public | entity_selectors                    | table    | otero
 public | entity_selectors_oid_seq            | sequence | otero
 public | fare_attributes                     | table    | otero
 public | fare_rules                          | table    | otero
 public | fare_rules_id_seq                   | sequence | otero
 public | feed_info                           | table    | otero
 public | frequencies                         | table    | otero
 public | geography_columns                   | view     | postgres
 public | geometry_columns                    | view     | postgres
 public | patterns                            | table    | otero
 public | route_type                          | table    | otero
 public | routes                              | table    | otero
:_
```

Figura 4.28: Tablas de la base de datos usada en el proyecto

Una vez que el modelado de la base de datos ha sido planteado, es necesario el estudiar cómo se realizará la comunicación entre los dispositivos y esta base. Por ello en el siguiente capítulo se estudiarán los protocolos de comunicación que la API utilizará para poder llevar la información desde los dispositivos hasta la base de datos.

```
 public | django_admin_log                | table    | otero
 public | django_admin_log_id_seq         | sequence | otero
 public | django_content_type             | table    | otero
 public | django_content_type_id_seq      | sequence | otero
 public | django_session                  | table    | otero
 public | django_site                     | table    | otero
 public | django_site_id_seq              | sequence | otero
 public | entity_selectors                | table    | otero
 public | entity_selectors_oid_seq        | sequence | otero
 public | fare_attributes                 | table    | otero
 public | fare_rules                      | table    | otero
 public | fare_rules_id_seq               | sequence | otero
 public | feed_info                       | table    | otero
 public | frequencies                     | table    | otero
 public | geography_columns               | view     | postgres
 public | geometry_columns                | view     | postgres
 public | patterns                        | table    | otero
 public | route_type                      | table    | otero
 public | routes                          | table    | otero
 public | shapes                          | table    | otero
 public | shapes_shape_pt_sequence_seq    | sequence | otero
 public | spatial_ref_sys                 | table    | postgres
 public | stop_feature_type               | table    | otero
 public | stop_features                   | table    | otero
 public | stop_features_id_seq            | sequence | otero
 public | stop_time_updates               | table    | otero
 public | stop_time_updates_oid_seq       | sequence | otero
 public | stop_times                      | table    | otero
 public | stop_times_stop_sequence_seq    | sequence | otero
 public | stops                           | table    | otero
 public | transfers                       | table    | otero
 public | transfers_id_seq                | sequence | otero
 public | trip_updates                    | table    | otero
 public | trip_updates_oid_seq            | sequence | otero
 public | trips                           | table    | otero
 public | universal_calendar              | table    | otero
 public | vehicle_positions               | table    | otero
 public | vehicle_positions_oid_seq       | sequence | otero
(56 rows)

(END)_
```

Figura 4.29: Tablas de la base de datos usada en el proyecto

```
tesis_development=> select agency_id,agency_name,agency_timezone,agency_url from agency;
 agency_id |              agency_name               |   agency_timezone   |            agency_url
-----------+----------------------------------------+---------------------+-------------------------------
 MB        | Metrobús                               | America/Mexico_City | http://www.metrobus.df.gob.mx/
 METRO     | Sistema de Transporte Colectivo-Metro  | America/Mexico_City | http://www.metro.df.gob.mx/
 RTP       | Red de Transporte de Pasajeros         | America/Mexico_City | http://www.rtp.gob.mx/
 STE       | Servicio de Transportes Eléctricos     | America/Mexico_City | http://www.ste.df.gob.mx/
 SUB       | Ferrocarriles Suburbanos               | America/Mexico_City | http://www.fsuburbanos.com/
(5 rows)

tesis_development=> _
```

Figura 4.30: Tabla Agency usada para la creación del archivo agency.txt

```
 service_id | monday | tuesday | wednesday | thursday | friday | saturday | sunday | start_date |  end_date
------------+--------+---------+-----------+----------+--------+----------+--------+------------+------------
 36238      | t      | t       | t         | t        | t      | f        | f      | 2013-08-29 | 2013-09-29
 38253      | t      | t       | t         | t        | t      | t        | t      | 2013-08-29 | 2013-09-29
 38742      | t      | t       | t         | t        | t      | f        | f      | 2013-08-29 | 2013-09-29
 38754      | t      | t       | t         | t        | t      | f        | f      | 2013-08-29 | 2013-09-29
 38824      | f      | f       | f         | f        | f      | t        | f      | 2013-08-29 | 2013-09-29
 38827      | f      | f       | f         | f        | f      | f        | t      | 2013-08-29 | 2013-09-29
 38832      | f      | f       | f         | f        | f      | t        | f      | 2013-08-29 | 2013-09-29
 38835      | f      | f       | f         | f        | f      | f        | t      | 2013-08-29 | 2013-09-29
 38881      | t      | t       | t         | t        | t      | f        | f      | 2013-08-29 | 2013-09-29
 36479      | t      | t       | t         | t        | t      | f        | f      | 2013-08-29 | 2013-09-29
 38501      | f      | f       | f         | f        | f      | t        | f      | 2013-08-29 | 2013-09-29
 38504      | f      | f       | f         | f        | f      | f        | t      | 2013-08-29 | 2013-09-29
 16233      | f      | f       | f         | f        | f      | f        | t      | 2013-08-29 | 2013-09-29
 36489      | f      | f       | f         | f        | f      | t        | f      | 2013-08-29 | 2013-09-29
 16092      | t      | t       | t         | t        | t      | f        | f      | 2013-08-29 | 2013-09-29
 39389      | f      | f       | f         | f        | f      | f        | t      | 2013-08-29 | 2013-09-29
 38274      | t      | t       | t         | t        | t      | t        | t      | 2013-08-29 | 2013-09-29
 26949      | f      | f       | f         | f        | f      | f        | t      | 2013-08-29 | 2013-09-29
 26656      | t      | t       | t         | t        | t      | f        | f      | 2013-08-29 | 2013-09-29
 14741      | t      | t       | t         | t        | t      | f        | f      | 2013-08-29 | 2013-09-29
 36384      | f      | f       | f         | f        | f      | t        | f      | 2013-08-29 | 2013-09-29
 28960      | f      | f       | f         | f        | f      | f        | t      | 2013-08-29 | 2013-09-29
 16203      | f      | f       | f         | f        | f      | t        | f      | 2013-08-29 | 2013-09-29
(23 rows)

(END)_
```

Figura 4.31: Tabla Calendar usada para la creación del archivo calendar.txt

trip_id	start_time	end_time	headway_secs	exact_times
36237	07:00:00	09:30:00	120	0
38880	05:00:00	24:30:00	180	0
38888	05:00:00	24:00:00	180	0
38895	05:00:00	24:30:00	180	0
38898	05:00:00	24:30:00	180	0
38904	06:00:00	22:00:00	240	0
38911	05:00:00	24:45:00	240	0
38962	05:00:00	24:00:00	300	0
38975	05:00:00	24:00:00	300	0
39280	05:00:00	24:00:00	240	0
39283	05:00:00	24:00:00	240	0
39344	05:00:00	23:30:00	300	0
39317	05:00:00	24:45:00	240	0
39337	05:00:00	23:30:00	300	0
38870	05:00:00	24:30:00	120	0
38741	05:00:00	24:00:00	240	0
38875	05:00:00	24:30:00	120	0
38811	06:00:00	24:00:00	300	0
38752	05:00:00	24:00:00	240	0
38877	05:00:00	24:30:00	120	0
38871	05:00:00	24:30:00	120	0
38891	05:00:00	24:00:00	240	0
38882	05:00:00	24:00:00	240	0
38899	05:00:00	24:30:00	240	0
38896	05:00:00	24:00:00	240	0
38979	06:00:00	24:00:00	360	0
38914	05:00:00	24:00:00	300	0
39281	05:45:00	24:00:00	300	0
38956	06:00:00	24:00:00	360	0
39285	06:00:00	24:00:00	360	0
39284	05:45:00	24:00:00	300	0
39346	05:00:00	23:30:00	360	0
38872	05:30:00	24:00:00	180	0
39318	05:00:00	24:00:00	300	0
39341	05:00:00	23:30:00	360	0
38884	05:30:00	23:30:00	300	0
38893	05:30:00	23:30:00	300	0
38826	07:00:00	24:00:00	360	0

:_

Figura 4.32: Tabla Frequencies usada para la creación del archivo frequencies.txt

```
 stop_id    |                                              stop_name
------------+------------------------------------------------------------
 STOP_14877 | El Caminero
 STOP_14878 | La Joya
 STOP_14879 | Santa Úrsula
 STOP_14880 | Fuentes Brotantes
 STOP_14881 | Ayuntamiento
 STOP_14882 | Corregidora
 STOP_14883 | Villa Olþmpica
 STOP_14884 | Perisur
 STOP_14885 | CCU
 STOP_14886 | Ciudad Universitaria
 STOP_14887 | Dr. Gálvez
 STOP_14888 | Bombilla
 STOP_14889 | Altavista
 STOP_14890 | Olivo
 STOP_14891 | Francia
 STOP_14892 | José Marþa Velasco
 STOP_14893 | Teatro de los Insurgentes
 STOP_14894 | Rþo Churubusco
 STOP_14895 | Felix Cuevas
 STOP_14896 | Parque Hundido
 STOP_14897 | Ciudad de los Deportes
 STOP_14898 | Colonia del Valle
 STOP_14899 | Nápoles
 STOP_14900 | Poliforum
 STOP_14901 | La Piedad
 STOP_14902 | Nuevo León
 STOP_14903 | Chilpancingo
 STOP_14904 | Campeche
 STOP_14905 | Sonora
 STOP_14906 | Álvaro Obregón
 STOP_14907 | Durango
 STOP_14908 | Glorieta Insurgentes
 STOP_14909 | Hamburgo
 STOP_14910 | Reforma
 STOP_14911 | Revolución
 STOP_14912 | Plaza de la República
 STOP_14913 | El Chopo
 STOP_14914 | Buenavista. Lþnea 1.
:_
```

Figura 4.33: Columnas de la tabla Stop utilizadas para crear el archivo stops.txt

stop_id	stop_lat	stop_lon	geom
STOP_14877	19.279392000	-99.169174000	0101000020E610000058552FBFD3CA58C098C3EE3B86473340
STOP_14878	19.280359000	-99.170097000	0101000020E6100000740987DEE2CA58C0A723809BC5473340
STOP_14879	19.283764000	-99.175598000	0101000020E61000008DCF64FF3CCB58C03BC8EBC1A4483340
STOP_14880	19.288380000	-99.174461000	0101000020E61000005B5B785E2ACB58C00BD28C45D3493340
STOP_14881	19.292501000	-99.177399000	0101000020E6100000F5D555815ACB58C01C0C7558E14A3340
STOP_14882	19.294146000	-99.181252000	0101000020E61000006A15FDA199CB58C0CD3FFA264D4B3340
STOP_14883	19.299402000	-99.185509000	0101000020E610000046072461DFCB58C0645B069CA54C3340
STOP_14884	19.304468000	-99.186066000	0101000020E610000071395E81E8CB58C0B4AD669DF14D3340
STOP_14885	19.314615000	-99.187408000	0101000020E610000090C01F7EFECB58C0BFD4CF9B8A503340
STOP_14886	19.322798000	-99.188431000	0101000020E61000005DA3E5400FCC58C0D636C5E3A2523340
STOP_14887	19.340818000	-99.189980000	0101000020E610000039B9DFA128CC58C05CE333D93F573340
STOP_14888	19.346811000	-99.187912000	0101000020E6100000A9A10DC006CC58C09FE40E9BC8583340
STOP_14889	19.351315000	-99.186630000	0101000020E6100000F59CF4BEF1CB58C02098A3C7EF593340
STOP_14890	19.354715000	-99.185272000	0101000020E61000005437177FDBCB58C090662C9ACE5A3340
STOP_14891	19.358360000	-99.184044000	0101000020E61000009A417C60C7CB58C0CF31207BBD5B3340
STOP_14892	19.361984000	-99.182716000	0101000020E61000002E1D739EB1CB58C0DEACC1FBAA5C3340
STOP_14893	19.365011000	-99.181747000	0101000020E61000005A492BBEA1CB58C025AE635C715D3340
STOP_14894	19.369080000	-99.180443000	0101000020E6100000B2F2CB608CCB58C0919BE1067C5E3340
STOP_14895	19.374292000	-99.178757000	0101000020E6100000973B33C170CB58C08927BB99D15F3340
STOP_14896	19.379322000	-99.177284000	0101000020E6100000A986FD9E58CB58C03FA7203F1B613340
STOP_14897	19.382601000	-99.176041000	0101000020E61000005470784144CB58C0BA6A9E23F2613340
STOP_14898	19.385506000	-99.175278000	0101000020E6100000556D37C137CB58C06A696E85B0623340
STOP_14899	19.389858000	-99.173683000	0101000020E6100000C1374D9F1DCB58C07F15E0BBCD633340
STOP_14900	19.393623000	-99.172462000	0101000020E610000060730E9E09CB58C013F4177AC4643340
STOP_14901	19.397873000	-99.171303000	0101000020E61000003AADDBA0F6CA58C01FF64201DB653340
STOP_14902	19.401871000	-99.169960000	0101000020E61000003468E89FE0CA58C0F8359204E1663340
STOP_14903	19.406485000	-99.168503000	0101000020E6100000C991CEC0C8CA58C08750A5660F683340
STOP_14904	19.409693000	-99.167320000	0101000020E6100000DFFDF15EB5CA58C06F66F4A3E1683340
STOP_14905	19.413143000	-99.166245000	0101000020E6100000E88711C2A3CA58C0439259BDC3693340
STOP_14906	19.416866000	-99.164856000	0101000020E610000029232E008DCA58C078D0ECBAB76A3340
STOP_14907	19.419903000	-99.164040000	0101000020E610000018CFA0A17FCA58C0077E54C37E6B3340
STOP_14908	19.423423512	-99.162468910	0101000020E6100000000000E465CA58C00030B87B656C3340
STOP_14909	19.427805000	-99.161057000	0101000020E6100000AEF204C24ECA58C0B610E4A0846D3340
STOP_14910	19.433065000	-99.158806000	0101000020E6100000261AA4E029CA58C0D00A0C59DD6E3340
STOP_14911	19.440411000	-99.155540000	0101000020E6100000124E0B5EF4C958C075CC79C6BE703340
STOP_14912	19.436152000	-99.157227000	0101000020E610000016C3D50110CA58C0C21550A8A76F3340
STOP_14913	19.443102000	-99.154594000	0101000020E61000001A8A3BDEE4C958C0CDCAF6216F713340
STOP_14914	19.446957000	-99.153030000	0101000020E6100000A453573ECBC958C0E1B721C66B723340

:_

Figura 4.34: Columnas en la tabla Stop.

route_id	service_id	trip_id	trip_headsign
ROUTE_18226	36479	38834	Indios Verdes - El Caminero
ROUTE_18226	36479	38836	El Caminero - Indios Verdes
ROUTE_18226	36479	38861	Indios Verdes - Dr. Gálvez
ROUTE_18226	36479	38865	Indios Verdes - Dr. Gálvez
ROUTE_18226	36479	38866	Indios Verdes - Dr. Gálvez
ROUTE_18226	36479	38867	Dr. Gálvez - Indios Verdes
ROUTE_18226	36479	38868	Dr. Gálvez - Indios Verdes
ROUTE_18226	36479	38869	Dr. Gálvez - Indios Verdes
ROUTE_18226	36479	38873	Indios Verdes - Glorieta de Los Insurgentes
ROUTE_18226	36479	38874	Glorieta de Los Insurgentes - Indios Verdes
ROUTE_18226	36479	38876	Buenavista 2 - El Caminero
ROUTE_18226	36479	38878	El Caminero - Buenavista 2
ROUTE_36644	36479	38883	Etiopía - Tenayuca
ROUTE_36644	36479	38885	Tenayuca - Etiopía
ROUTE_36644	36479	38886	Tenayuca - Buenavista 3
ROUTE_36644	36479	38887	Buenavista 3 - Tenayuca
ROUTE_36644	36479	38889	Tenayuca - La Raza
ROUTE_36644	36479	38890	La Raza - Tenayuca
ROUTE_36644	36479	38892	Tenayuca - Balderas
ROUTE_36644	36479	38894	Balderas - Tenayuca
ROUTE_36644	38501	38900	Tenayuca - Etiopía
ROUTE_36644	38501	38902	Tenayuca - Etiopía
ROUTE_36644	38501	38903	Tenayuca - Etiopía
ROUTE_36644	38501	38905	Etiopía - Tenayuca
ROUTE_36644	38501	38906	Etiopía - Tenayuca
ROUTE_36644	38501	38907	Etiopía - Tenayuca
ROUTE_36644	38501	38908	Tenayuca - Buenavista 3
ROUTE_36644	38501	38909	Tenayuca - Buenavista 3
ROUTE_36644	38501	38910	Tenayuca - Buenavista 3
ROUTE_36644	38501	38912	Buenavista 3 - Tenayuca
ROUTE_36644	38501	38913	Buenavista 3 - Tenayuca
ROUTE_36644	38501	38915	Buenavista 3 - Tenayuca
ROUTE_36644	38501	38916	Tenayuca - La Raza
ROUTE_36644	38501	38917	Tenayuca - La Raza
ROUTE_36644	38501	38918	La Raza - Tenayuca
ROUTE_36644	38501	38919	La Raza - Tenayuca
ROUTE_36644	38504	38927	Tenayuca - Etiopía
ROUTE_36644	38504	38929	Tenayuca - Etiopía

:_

Figura 4.35: Tabla Trips usada para la creación del archivo trips.txt

id	from_stop_id	to_stop_id	transfer_type	min_transfer_time
1	STOP_14139	STOP_14157	0	
2	STOP_14162	STOP_14141	0	
3	STOP_14109	STOP_14058	0	
4	STOP_14155	STOP_14145	0	
5	STOP_14159	STOP_14155	0	
6	STOP_14159	STOP_14145	0	
7	STOP_14190	STOP_14187	0	
8	STOP_14169	STOP_14165	0	
9	STOP_14166	STOP_14164	0	
10	STOP_14168	STOP_14167	0	
11	STOP_14175	STOP_14144	0	
12	STOP_14103	STOP_14099	0	
13	STOP_14182	STOP_14051	0	
14	STOP_14226	STOP_14216	0	
15	STOP_14214	STOP_14226	0	
16	STOP_14214	STOP_14216	0	
17	STOP_14215	STOP_14214	0	
18	STOP_14215	STOP_14226	0	
19	STOP_14137	STOP_14161	0	
20	STOP_14178	STOP_14064	0	
21	STOP_14112	STOP_14108	0	
22	STOP_14062	STOP_14054	0	
23	STOP_14122	STOP_14062	0	
24	STOP_14122	STOP_14054	0	
25	STOP_14157	STOP_14139	0	
26	STOP_14141	STOP_14162	0	
27	STOP_14058	STOP_14109	0	
28	STOP_14145	STOP_14155	0	
29	STOP_14155	STOP_14159	0	
30	STOP_14145	STOP_14159	0	
31	STOP_14187	STOP_14190	0	
32	STOP_14165	STOP_14169	0	
33	STOP_14164	STOP_14166	0	
34	STOP_14167	STOP_14168	0	
35	STOP_14144	STOP_14175	0	
36	STOP_14099	STOP_14103	0	
37	STOP_14051	STOP_14182	0	
38	STOP_14216	STOP_14226	0	

:_

Figura 4.36: Tabla Transfer usada para la creación del archivo transfer.txt

Capítulo 5

Estándares y protocolos

En el último capítulo de éste trabajo se describirán los protocolos a utilizar y se darán la razones del porqué han sido elegidos sobre otras opciones disponibles. Se describirán arquitecturas como REST y se hablará de protocolos de representación de información como JSON o YAML.

5.1. Introducción a la arquitectura REST

En el año de 1993 se desarrolló lo que es conocido como el estilo arquitectónico de la web. Nacido por el temor que existía de que la expansión de internet en el mundo pudiese generar una sobrecarga sobre los sistemas web y que estos no pudiesen escalar fácilmente, permitió la expansión de nuevas empresas y estándares que actualmente se utilizan en todo el mundo.

Se descubrió que existían algunas restricciones que generaban el problema de la saturación y sus soluciones fueron divididas en 6 categorías las cuales son referidas como el estilo arquitectónico de la web:

1. Sistemas cliente servidor
2. Interfaces uniformes
3. Sistemas basados en capas
4. Sistemas de Cache
5. Sistemas sin estado
6. Sistemas de código bajo demanda

A partir de los inicios del siglo XXI y después de la crisis que provocó la burbuja de las punto-com en las empresas tecnológicas, comenzaron investigaciones a nivel global para el desarrollo de nuevos sistemas computacionales que pudiesen aprovechar la infraestructura de internet. El doctor Fielding en su disertación doctoral nombró y describió la arquitectura web mencionada arriba como “Representational State Transfer(REST)” y ha sido utilizada ampliamente en la

industria tomando mayor fuerza a partir de 2004 cuando el número de APIs en el mundo comenzó a crecer.

5.1.1. Sistemas cliente servidor

Esta solución implica la separación de los trabajos, necesidades y roles en la comunicación entre los sistemas Web y los clientes. Esto permitió crear sistemas que fuesen independientes no sólo en su comportamiento, también en el lenguaje de programación en el que son diseñados, la arquitectura, las tasas de transmisión que pueden manejar, el procesamiento de cada uno, etc, siempre y cuando cumplan con las interfaces uniformes de la web.

En este trabajo se utiliza ésta arquitectura de comunicación que permitió el diseño de un receptor independiente de los dispositivos de seguimiento de vehículos. Al mismo tiempo también permitió la división de la administración del sistema, por lo que los administradores de los servidores no deben ser necesariamente los mismos administradores de los dispositivos de localización.

5.1.2. Interfaces Uniformes

Las interacciones entre los componentes web en internet (tanto clientes, como servidores y sistemas de red intermedios) dependen de la uniformidad de las interfaces. Si alguno de los componentes no cumple con alguna de estas interfaces, la comunicación dentro de internet y la web se vuelve imposible.

Los componentes de la web interactúan entre ellos consistentemente con interfaces que se componen principalmente de cuatro categorías:

1. **Identificación de recursos:** Cada uno de los recursos que se encuentran en la web pueden ser alcanzados a través de un identificador único. Por ejemplo, una página web puede ser alcanzada a través de su URI como htttp://www.unam.mx o un router puede ser alcanzado a través de su dirección IP, cada uno identifica únicamente el recurso específico.

2. **Manipulación de recursos a través de de representaciones:** Los clientes pueden modificar la representación de los recursos. El mismo recurso puede ser reenviado a diferentes clientes de diferentes maneras. Por ejemplo, una página puede ser enviada como HTML para un navegador web, mientras que es enviada como XML o JSON para un sistema autónomo. La idea principal es que la representación del recurso es sólo una manera de trabajar con él, sin necesidad de ser el propio recurso. Este concepto permite presentar recursos en diferentes formatos sin necesidad de cambiar el identificador.

3. **Mensajes auto descriptivos:** El estado deseado de un recurso puede ser presentado en el protocolo de petición del cliente. Y el recurso en estado actual puede ser presentado como respuesta a este mensaje desde el servidor. Un ejemplo es un router al cual se le

quiere modificar una configuración. El mensaje que se envía sugiere una actualización de una de las configuraciones del router, es cuestión del dispositivo y sus políticas de seguridad si permite esta modificación de su configuración.

Los mensajes contienen metadatos (cabeceras para paquetes a nivel de red) para entregar información adicional, la representación del formato y el tamaño, así como información del cuerpo del mensaje. Un mensaje HTTP provee cabeceras para organizar varios tipos de metadatos dentro de campos que siguen un estándar.

4. **Estado de la aplicación como un motor de enlaces:** La representación de los recursos puede incluir ligas hacia recursos relacionados que permiten navegar a través de la información y las aplicaciones de una manera directa.

5.1.3. Sistemas basados en capas

Los sistemas basados en capas permiten que los intermediarios en las redes como proxies o puertas de enlace sean programados y configurados de una forma transparente para los usuarios de la web. Hablando de una forma general, los sistemas intermediarios interceptan la comunicación entre el cliente y el servidor para realizar tareas específicas en algunas capas sin necesidad de intervención del usuario o de metadatos adicionales.

5.1.4. Sistemas de Cache

Este tipo de sistemas son uno de los más importantes en la arquitectura de los servicios web ya que permiten tener información de respuesta dentro de memorias de rápido acceso distribuidas a lo largo de la red. Esto reduce la latencia que existe para un usuario, incrementa la disponibilidad y confiabilidad de los sistemas y elimina carga sobre los servidores reduciendo el costo de los servicios web.

Al poder existir en cualquier punto de la red entre el cliente y el servidor, pueden ser parte de la propia organización de un servidor web con contenido especial guardado en lo que se conoce como redes de deliberación de contenido (CDN[1]).

5.1.5. Sistemas sin estado

Este tipo de solución propone que ni los dispositivos web ni los servidores necesitan memorizar el estado de las aplicaciones de sus clientes. Por ende cada uno de los clientes debe de incluir toda la información necesaria en cada una de las interacciones con el servidor. Esto permite atender a una mayor cantidad de clientes al mismo tiempo, lo que ayuda a escalar más fácilmente.

[1]Content Delivery Networks: Red de distribución de contenido de gran escala constituido de múltiples servidores en centros de datos a través de internet. Su objetivo es entregar contenido a los usuarios finales en el menor tiempo posible.

5.1.6. Sistemas de código bajo demanda

Este tipo de subsistemas permiten enviar programas ejecutables como scripts o plug-ins a clientes.

Obliga a crear tecnologías que permitan a cada uno de los clientes, no importando su naturaleza, entender y ejecutar el código que descargan desde los servidores. Es por ello que estos sistemas son los únicos dentro de la arquitectura web que son opcionales.

5.1.7. API con arquitectura REST

Los servicios web generalmente son sistemas de propósito específico que intentan resolver un problema. Los clientes utilizan interfaces de programación de aplicaciones (API[2]) para comunicarse con esos servicios. De una manera general se puede decir que las API permiten acceder a información y funciones dentro del servidores que facilitan la interacción entre computadoras, por lo que una API se puede resumir como una computadora esperando peticiones para dar recursos a un cliente. En la figura 5.1 se puede ver un diagrama simplificado de cómo funciona una API.

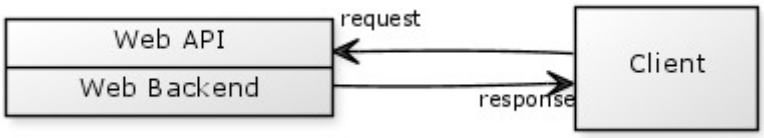

Figura 5.1: Simplificación del funcionamiento de una API

5.2. Diseño de URI para la obtención de recursos

Las API con arquitectura REST utilizan URI para el acceso a los recursos. Es por ello que se debe de poner mucho empeño en el diseño de estas para hacer que la API sea sencilla de utilizar.

Los usuarios deben de tomar las URI como identificadores opacos, por lo que no deben de necesitar ver la URL en busca de otro tipo de información más allá de qué recurso representa. El formato de la URL que se maneja en la mayoría de los servicios web tiene el siguiente formato:

URI = protocolo :// nombre.de.dominio/ruta/al/recurso[? variables][# fragmentos]

En donde la diagonal (/) es traducida como la relación jerárquica que existe en los recursos dentro del servidor y no se puede utilizar en el nombre de los recursos. También se debe de tomar en cuenta que cuando se encuentra al final de una URL no añade contenido semántico por lo que se trata igual las siguientes dos URL:

[2]Application Programming Interfaces

1. http://.../api/traffic

2. http://.../api/traffic/

Sin embargo esto es considerado una mala práctica por los libros dedicados al tema de diseño de servicios web [5] [6] [7] en donde se establece que lo correcto es hacer que dos URI diferente envíen a recursos diferentes, sin permitir a los usuarios el detectar el mismo recurso a través de diferentes caminos. En su lugar se debe responder con el estatus *301 "Moved Permanently"*.

La URL tampoco debe incluir la extensión o representación del recurso. Aunque esto es común en API privadas y que tienen muchas representaciones, la manera en cómo esto debe ser manejado es a través de la cabecera MediaType del protocolo HTTP.

Cuando se diseñan las URL se pueden manejar patrones estándares establecidos que se mantengan en un futuro. De esta manera se pueden comunicar consistentemente los clientes con los servidores. Para realiza estos patrones podemos dividir los recursos en 4 diferentes arqueo-tipos

1. **Documentos:** Es un concepto singular que puede estar relacionado con un objeto o un registro en una base de datos. Este tipo de recurso puede contener ligas hacia otros recursos, así como sus valores propios. Esta estructura mínima es la base de otro tipo de arqueo-tipos que se usan. Un ejemplo de un recurso que se comporta como un documento sería:

   http://www..../api/vehicle/plate/1250258

   El cual representa la información del vehículo con placa 1250258.

2. **Colecciones:** Dentro de una API una colección se puede entender como un conjunto de documentos o un directorio de documentos. Esta palabra no debe ser confundida con las colecciones que se encuentran dentro de los lenguajes de programación ya que propiamente hablando, éstos últimos son una estructura de datos que se encuentran contiguos dentro de la memoria.

   Es posible programar la API para permitir a los usuarios el aumentar la información que es presentada cuando se llama por este recurso aunque esto es opcional para el programador.

   Un ejemplo de una colección se muestra a continuación:

   http://.../vehicles
   http://.../vehicles/taxis

3. **Sistemas de almacenado:** Las URL que apuntan a sistemas que son manejados por el cliente son conocidas como sistemas de almacenado. Este tipo de recursos permiten al cliente trabajar en recursos propios. Por sí mismos, estos recursos no pueden generar

otros documentos, por lo que nunca se crean nuevas URL a partir de peticiones a ellos. En lugar de ello, el cliente utiliza una URL que identifica sus recursos a través de su ID y que le permite acceder a sus nuevos documentos a través de variables.

A continuación se muestra un ejemplo de una URL para un sistema de almacenado:

PUT http://.../users/256?amigo='juan'

4. **Controlador:** Es un documento que modela una acción que pueden verse como métodos en un lenguaje de programación, a los cuales se le puede enviar argumentos y retorna variables.

   Este tipo de acciones están pensadas para poder realizar efectos sobre los datos que no estén contemplados en las cabeceras GET, POST, PUT o DELETE del protocolo HTTP.

   Generalmente aparecen al final de la URL sin recursos de jerarquías más bajas y con sus respectivos argumentos opcionales. Por ejemplo la siguiente URI podría representar el reenvío de un correo a un usuario:

   POST http://.../users/256/emails/4/resend

Las siguientes reglas se sugieren a la hora de diseñar las URL del proyecto y que serán utilizadas en el prototipo:[3]

1. **Los nombres de los documentos deben ser sustantivos:** Todas las palabras que distingan a los documentos deben ser nombrados utilizando un sustantivo singular que las identifique:

   a) http://.../usuarios/juan

2. **Los nombres plurales deben ser utilizados para colecciones:** Una colección debe utilizar sustantivos en plural que identifiquen de la manera más general el tipo de recursos. Esto implica que un buen diseño de URI siempre mantendrá colección con relaciones semánticas entre sus documentos.

   Las siguientes URL representan distintas colecciones:

   a) http://.../vehicles (Representa la colección donde están los vehículos)

   b) http://.../users (Representa la colección de usuarios)

3. **Las sistemas de almacenado deben estar en plural y tener un identificador en su jerarquía:** Cuando se representa este tipo de recurso es necesario establecer en la URL de quién son los datos, así como los datos que se van a utilizar. También es

[3]Estas recomendaciones están basadas en algunas API que actualmente existen a nivel comercial. En general estas reglas siempre son respetadas no importando la aplicación.

importante nombrar con sustantivos en plural que identifiquen de la manera más general las colecciones que se están buscando.

Las siguientes URL muestran distintos sistemas de almacenado:

http://.../users/256/options
http://.../devices/32/vehicles

4. **Para los controladores se debe utilizar un verbo:** Al igual que esta regla se utiliza para funciones o métodos dentro de los lenguajes de programación, el poner el nombre de la acción que realiza el controlador es considerado como una buena práctica.

   Las siguientes URL muestran distintos sistemas de almacenado:

   http://.../users/256/reset-password
   http://.../users/256/send-confirmation-email
   http://.../vehicle/91/send-alert

5. **Las URL que no son fijas deben de incluir los valores de las variables:** Algunas URL tienen jerarquías que dependen de variables identificando con la misma estructura de la URL diferentes documentos o colecciones. Un ejemplo de esto es:

   http://www.ejemplo.com/deportes/Id deporte/equipos/Id equipo/jugadores/Id jugador

   Esto se conoce como estructura de sintaxis de la URL en donde las palabras entre llaves representan variables que son sustituidas dependiendo del recurso que se busque pero que la estructura en su totalidad puede representar múltiples recursos.

6. **Los verbos HTTP no deben de ser utilizados dentro de las URL:** Las URL no deben de indicar que verbo HTTP debe usarse para poder mandar a llamar el recurso o para realizar una acción. En lugar de ello debe indicarse en la documentación el verbo HTTP con el que se debe mandar a llamar la URL y obtener con ello URL "limpias".

   Un ejemplo de URL mal diseñadas son las siguientes donde se intenta eliminar el usuario con el id 256:

   a) DELETE http://.../users/deleteUser?id=256
   b) DELETE http://.../users/deleteUser/256
   c) DELETE http://.../users/256/deleteUser

   La URL correcta para realizar esta acción es:

   DELETE http://.../users/256

7. **Las variables en la URL son utilizadas para filtrar información, no para enviar información al servidor:** Aunque es posible enviar información al servidor a través de la URL esto no es recomendado si esta información realizará alguna modificación en los servidores. Sin embargo, estas variables pueden ser utilizadas para mandar un criterio de búsqueda o filtrar información en una colección. Por ejemplo las URL:

   a) GET http://.../users

   b) GET http://.../users?role=admin

   Ambas apuntan a la colección de usuarios, la diferencia es que (1) obtiene información de todos los usuarios en el sistema mientras que (2) sólo de aquellos que tienen permisos de administrador.

Mientras que las reglas presentadas aquí no son las únicas que se deben utilizar, sí son las más básicas y las que se pueden considerar como necesarias para tener una API compatible con los estándares actuales. Otra cuestión importante que se debe de analizar cuando se crea una API son los metadatos, estos son las cabeceras e información adicional que es enviada en cada paquete HTTP para poder trabajar con la API.

5.3. Introducción al protocolo HTTP

La mayoría de los servidores, aplicaciones y servicios web utilizan el protocolo HTTP para comunicarse. Es considerado como uno de los protocolos más comunes de internet y por ello tiene un amplio soporte.

En esta sección se intenta entender un poco este protocolo para poder aplicarlo en el proyecto y en el diseño de la API.

5.3.1. HTTP: Sistema de multimedia de Internet

Millones de documentos HTML, archivos de texto, imágenes, películas, archivos de audio, entre otros, son transportados a través del Internet diariamente. El protocolo HTTP transporta parte de esta información al rededor del mundo hasta los navegadores de los usuarios.

Debido a que HTTP puede utilizar protocolos seguros de transmisión de datos, asegura que tu información no será dañada ni modificada durante la transmisión no importando desde donde provenga. Esto permite el acceso a la información sin necesidad de preocuparse de la integridad.

5.3.2. Mensajes HTTP

Si se puede entender al protocolo HTTP como uno de los que mantienen en movimiento el flujo de información a través de internet, entonces los mensajes HTTP se pueden entender como los

paquetes que permiten administrar este flujo. Una transacción HTTP consiste de un comando de pedido (mandado desde el cliente hacia el servidor y un resultado de respuestas (desde el servidor hasta el cliente).

Los mensajes HTTP son bloques de información que son enviados entre aplicaciones que utilizan el protocolo HTTP. Estos bloques de datos comienzan con meta información que describe el contexto del mensaje, seguido de información opcional. Estos mensajes fluyen entre clientes, servidores y proxies.

La especificación HTTP [43] define un conjunto común de métodos de petición o mensajes. Por ejemplo, el método GET pide un documento a un servidor, mientras que el método POST envía información que será procesada por el servidor para realizar alguna operación.

No todos los métodos deben de ser implementados, por ejemplo la versión 1.1 del protocolo sólo establece que un servidor debe implementar GET y HEAD dejando todos los demás métodos como opcionales. Inclusive cuando los servidores implementan todos probablemente algunos tengan uso restringidos. Por ejemplo, servidores que soporten DELETE y PUT no permitirán a cualquiera eliminar, modificar o crear recursos. En la tabla 5.1 podemos ver los métodos más comunes utilizados en internet:

Método	**Descripción**	**Cuerpo**
GET	Obtiene un recurso desde un servidor	No
HEAD	Obtiene sólo las cabeceras de un recurso desde el servidor	No
POST	Envía información al servidor para ser procesada	Sí
PUT	Envía información al servidor para actualizar recursos	Sí
TRACE	Crea el recorrido a través de proxies hasta el servidor	No
OPTIONS	Determina qué método puede operar en un servidor	No
DELETE	Elimina un recurso del servidor	No

Tabla 5.1: Métodos HTTP más utilizados en internet

1. **GET:** Es el método más comúnmente utilizado para pedir recursos a un servidor. Es obligatoria su implementación para la versión 1.1 de HTTP. En la figura 5.2 se puede ver un ejemplo de una comunicación con este tipo de mensaje.

2. **HEAD:** El método HEAD se comporta de una manera similar al método GET excepto que sólo produce cabeceras. Permite al cliente inspeccionar cabeceras para un recurso sin necesidad de obtener el recurso reduciendo el consumo de ancho de banda. Utilizando HEAD es posible:

   I. Encontrar algún tipo de información sobre un recurso sin obtenerlo de verdad

   II. Ver si existe un objeto mirando el código de estado en la respuesta

   III. Testear si un recurso ha sido modificado mirando las cabeceras.

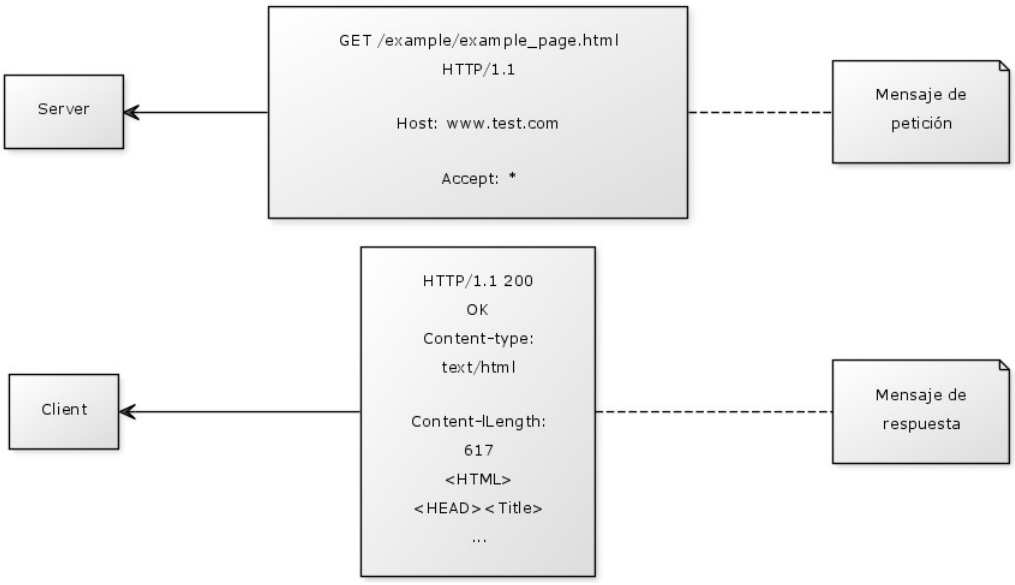

Figura 5.2: Ejemplo de mensaje GET

Los desarrolladores deben de asegurarse de que el método HEAD retorne las mismas cabeceras que retornaría el método GET. Este método es uno de los requeridos por la versión 1.1 del protocolo.

En la figura 5.3 se puede ver un ejemplo de comunicación con este mensaje.

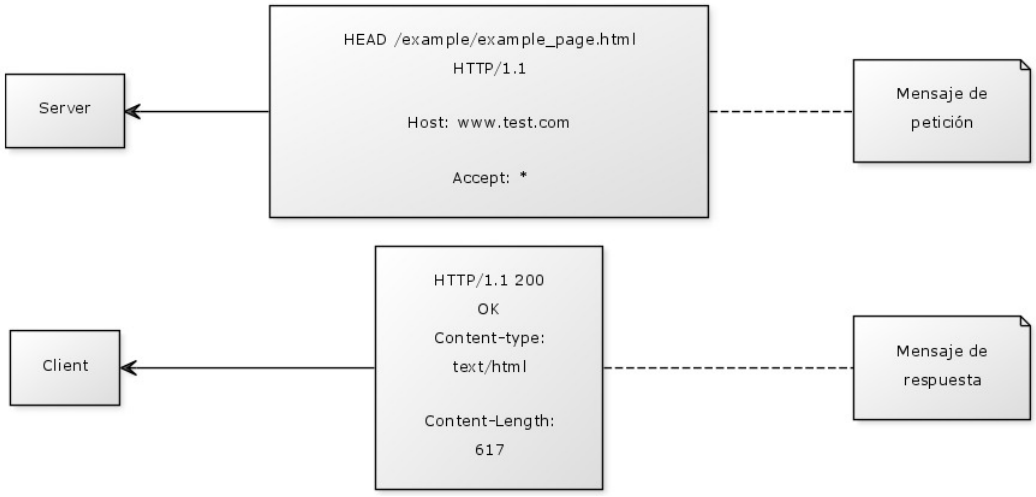

Figura 5.3: Ejemplo de mensaje HEAD

3. **PUT:** El método PUT originalmente fue diseñado para escribir documentos en los servidores, aunque actualmente es principalmente utilizado para actualizar documentos en los servidores. Su función es la inversa de lo que el método GET hace en los documentos al subir recursos al servidor en lugar de descargarlos.

   En la figura 5.4 se muestra cómo se ve la comunicación utilizando este método. La semántica del método PUT está diseñada para que el servidor tome lo que se encuentre en el cuerpo y con ello cree un nuevo documento o actualice uno ya existente.

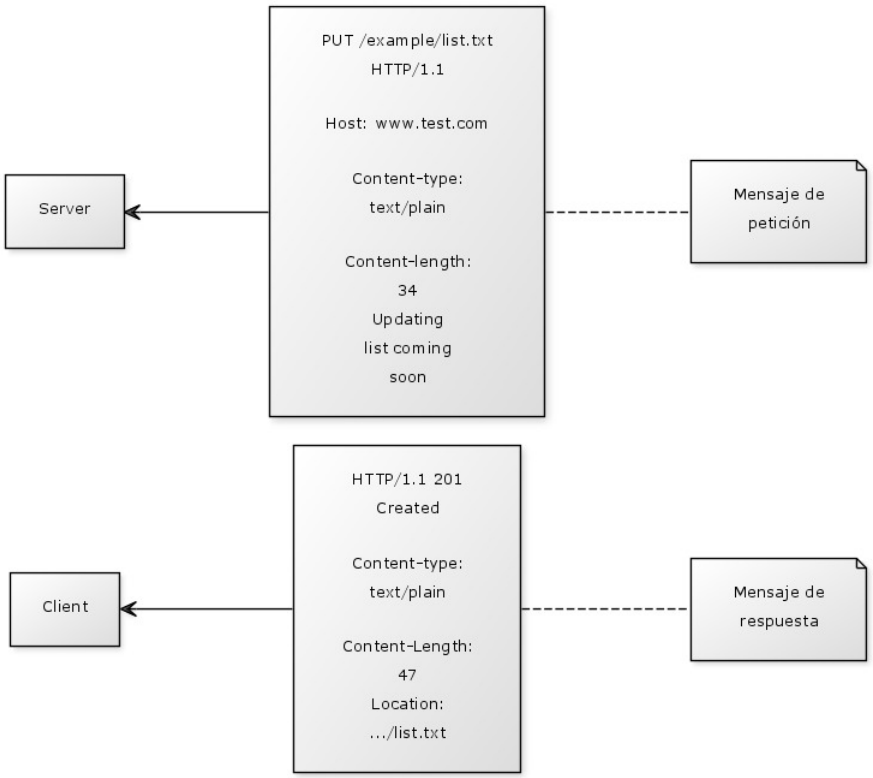

Figura 5.4: Ejemplo de mensaje PUT

4. **POST:** El método POST fue diseñado para enviar información al servidor sin forzar al servidor a guardar esa información como en el método PUT.

   La información de formularios generalmente es enviada al servidor que utiliza esta información para realizar una operación especial como designar una sesión u obtener una lista.

   En la figura 5.5 se muestra cómo se ve la comunicación para este tipo de mensaje

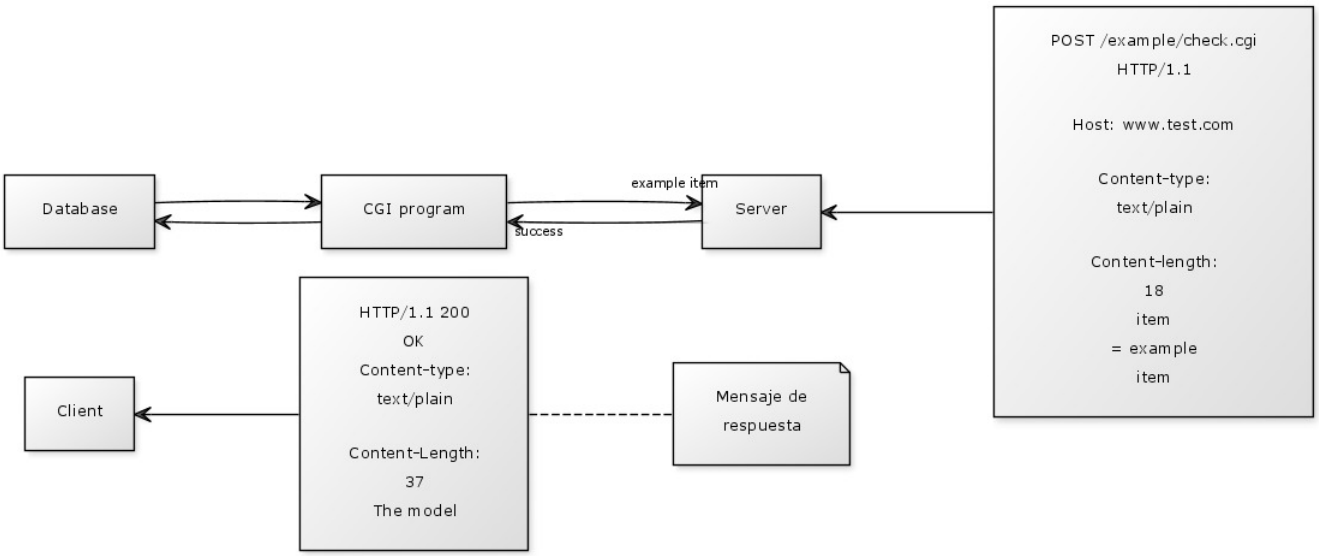

Figura 5.5: Ejemplo de mensaje POST

5. **TRACE:** Cuando un cliente realiza una petición, ésta debe viajar a través de diferentes dispositivos y sistemas como firewalls, proxies, gateways y otras aplicaciones. Cada uno de estas tiene la oportunidad de modificar la petición HTTP original. El método TRACE permite a los clientes el ver cómo se ve la petición cuando finalmente llegue al servidor.

   Una petición TRACE inicia un bucle de diagnóstico en el servidor de destino. El servidor envía al final de la petición un mensaje TRACE de retorno, con el mensaje de petición que recibió como respuesta. De esta manera el cliente puede ver cuál es la cadena de dispositivos a través de los que pasa el paquete.

   Este método es utilizado principalmente para diagnósticos, ya que puede verificar que las peticiones vayan a través de un camino seguro o preestablecido. También sirve para ver los efectos que tienen estos dispositivos en las peticiones. En la figura 5.6 se muestra cómo se ve la comunicación para este tipo de mensaje

6. **OPTIONS:** El método OPTIONS realiza una pregunta al servidor acerca de qué capacidades están soportadas. Es posible preguntarle qué métodos HTTP soporta en general para algunos recursos.

   Esto provee información para el cliente que le permite determina la mejor vía de acceso para los recursos que este sistema tiene.

   En la figura 5.7 se puede ver un ejemplo de un paquete usando este método

7. **DELETE:** El método DELETE, como su nombre lo indica, le pide al servidor el eliminar ciertos recursos. Sin embargo es posible que el recurso no sea eliminado ya que el protocolo permite al servidor sobre escribir el método sin necesidad de decirle al cliente. En la figura 5.8 se puede ver un ejemplo de este método

Los mensajes HTTP son relativamente simples. En la figura 5.9 se muestra una simplificación del paquete. Cada paquete consiste de tres partes:

1. Una linea de inicio describiendo el mensaje.
2. Un bloque de cabeceras que contienen los atributos.
3. Un cuerpo que contiene los datos.

Las lineas de inicio y las cabeceras son texto en ASCII divididas solamente en diferentes lineas. Cada linea termina con una secuencia de dos caracteres: un retorno de carro y de una nueva linea. Esta secuencia es conocida como CRLF[4]. Es necesario establecer que mientras la especificación HTTP establece específicamente secuencias de fin de linea CRLF, aplicaciones robustas deben de permitir sólo nueva linea.

[4]CRLF: Carriage return and Line Feed

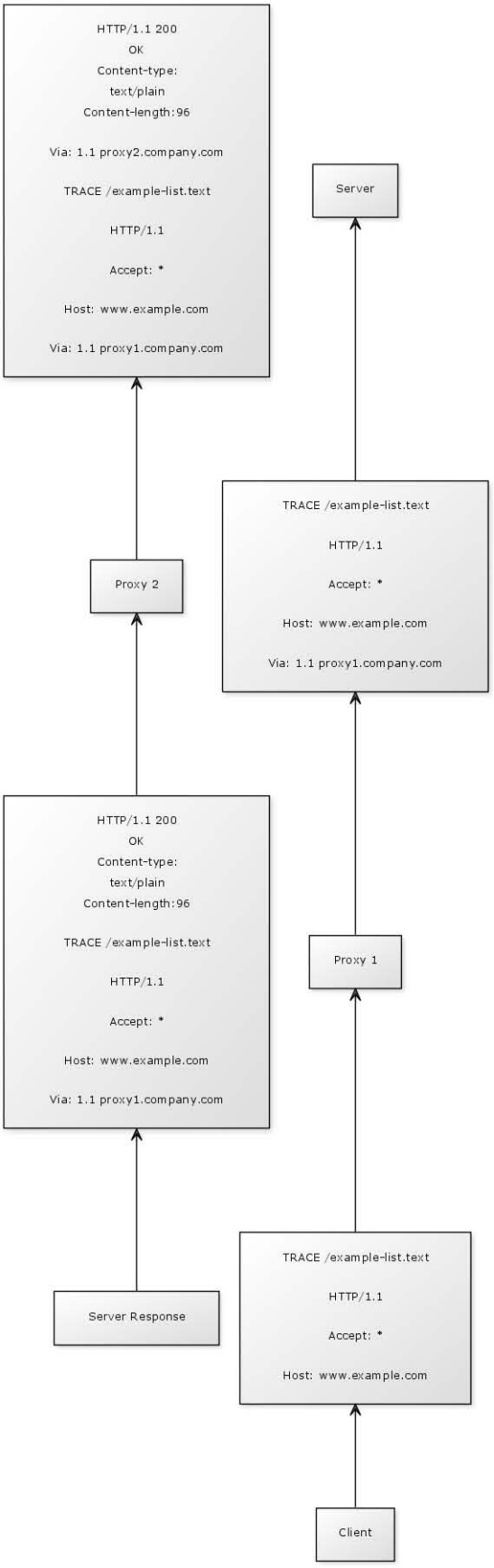

Figura 5.6: Ejemplo de mensaje TRACE

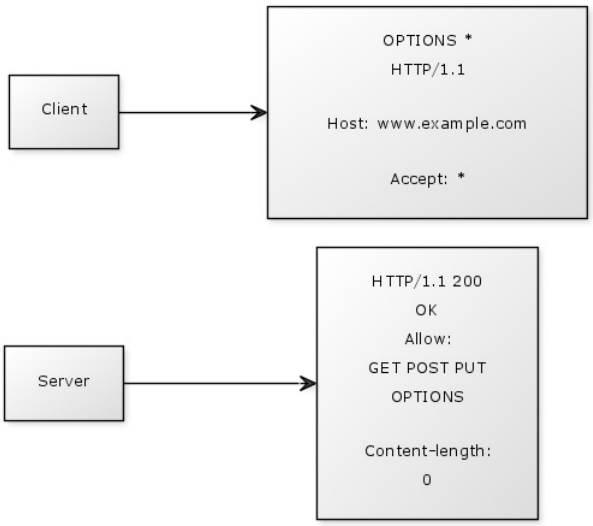

Figura 5.7: Ejemplo de mensaje OPTIONS

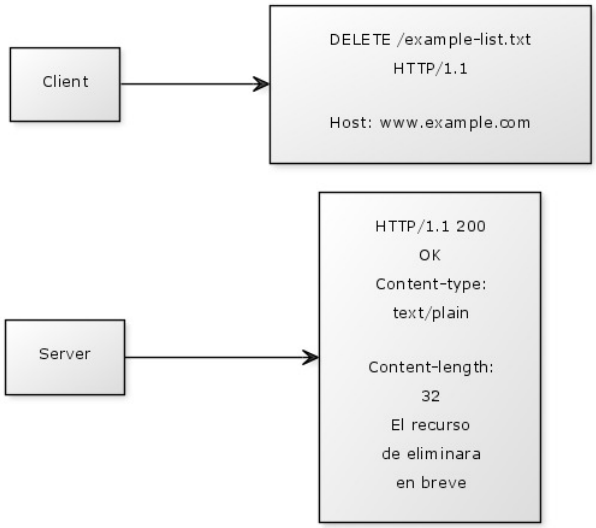

Figura 5.8: Ejemplo de mensaje DELETE

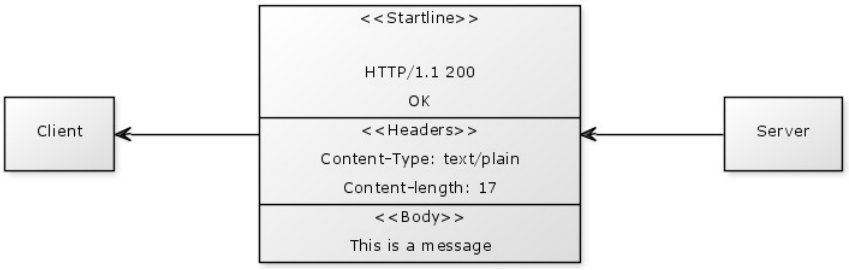

Figura 5.9: Esquema simplificado del protocolo HTTP

El cuerpo es un conjunto opcional de datos provenientes de capas inferiores en el modelo OSI. Al contrario de los inicios de linea y las cabeceras, el cuerpo puede contener texto, información binaria o puede estar vacío. La cabecera contiene información acerca del cuerpo del mensaje como el tipo de información que contiene (MIME type[5])

[5]Multipurpose Internet Mail Extensions: fueron originalmente diseñados para el intercambio de correos entre

5.3.3. Métodos HTTP Seguros

El estándar también define lo que es considerado como métodos seguros. Los métodos GET y HEAD son seguros lo que implica que no realizan ninguna operación sobre el recurso que lo pueda alterar.

No existe ninguna garantía de que un recurso seguro no realizará una acción sobre un recurso que pueda llegar a alterarlo. Los métodos seguros están pensados para permitir a los desarrolladores hacer saber a los clientes cuándo se han implementado también métodos inseguros.

5.3.4. Sintaxis del mensaje

Todos los métodos HTTP pueden ser categorizados en dos tipo: métodos de petición o métodos de respuesta. Los métodos de petición son todos aquellos que se hacen desde un cliente hacia un servidor. Todos los mensajes de respuestas son aquellos que son enviados desde el servidor hacia el cliente después de que un mensaje de petición ha sido recibido. Ambos tienen la misma estructura básica.

La estructura básica de un mensaje de petición sigue la siguiente forma:

<método><URL><versión>
<cabeceras>
<cuerpo del paquete>

La estructura básica del mensaje de respuesta es la siguiente (como se puede ver los mensajes sólo difieren en las lineas de inicio):

<versión><código de estado><frase con la razón>
<cabeceras>
<cuerpo del paquete>

1. **Método:** Es la acción que el cliente quiere realizar en los recursos del servidor. Es una palabra que representa el mensaje HTTP que se requiere.

2. **URL:** Es el identificador único que establece en qué lugar de internet y del servidor se encuentra el recurso que se está pidiendo. Si se está hablando con un servidor directamente, entonces esto refleja el camino absoluto dentro del servidor hacia el recurso.

3. **versión:** La versión del protocolo que se está utilizando en la comunicación. El formato es:

HTTP/<versión mayor>.<versión menor>

empresas de correo electrónico, pero después fueron adoptadas por el estándar HTTP

En donde la versión mayor y menor son ambos enteros.

4. **Código de estado:** Un número de 3 dígitos que describe qué es lo que ha sucedido dentro del servidor. El primer dígito de cada uno de los códigos describe estado general de la clase: éxito, error, no encontrado, etc. El estándar describe una lista exclusiva de códigos y su significado los cuales son:

   a) 100-199 Códigos de información: Son códigos relativamente nuevos y aún generan controversia acerca de la mejor forma de utilizarlos. Por ejemplo el código 100 está hecho para que un cliente envíe un paquete al servidor con información, sólo la para ver si el servidor la aceptaría.

   b) 200-299 Códigos de éxito: Cuando una petición es exitosa, la respuesta se envía con un código en éste rango.

   c) 300-399 Códigos de redireccionamiento: Estos códigos le dicen a los clientes la nueva localización de los recursos o proveen una respuesta alternativa.

   d) 400-499 Códigos de error de cliente: Cuando una petición que un servidor no puede manejar es enviada, éste regresa un paquete explicando la situación. Entre los problemas que pueden ocurrir son:

      1) El recurso no fue encontrado
      2) Paquete HTTP con mal formato
      3) El recurso no existe dentro de esa URI
      4) El método HTTP no es soportado

   e) 500-599 Códigos de error de servidor: Cuando existe un petición correcta que genera un error dentro del servidor, éste responde con un paquete indicando que un error interno ha causado que no se pueda procesar. El error se puede deber a:

      1) Limitación en el servidor de recursos
      2) El servicio pedido no está funcionando
      3) Un error fatal a ocurrido en el servidor
      4) EL servidor no soporta esa versión HTTP

5. **Frase de respuesta:** Una versión leíble para humanos que consiste en un texto completo haciendo referencia al código de estado. Las frases que se relacionan con cada código de estado están definidas en el estándar HTTP. [43]

   Este mensaje está pensado solamente para consumo humano, por lo que lineas “HTTP/1.1 200 SUCCESS” y “HTTP/1.1 200 OK” deben de ser tratadas como equivalentes.

6. **Cabeceras:** Cero o más cabeceras, cada una con un nombre distinto, seguidas de dos puntos (:), un espacio en blanco opcional, el valor de la cabecera y finalmente un caracter final CRLF.

Las cabeceras deben de ser terminadas por una linea blanca, haciendo el final de la lista de cabeceras el inicio del cuerpo. Algunas versiones de HTTP, como 1.1 requieren que ciertas cabeceras se encuentren presentes para la petición y la respuesta para ser consideradas válidas.

7. **Cuerpo:** Contiene un bloque de información creada en capas del modelo OSI anteriores. No todos los mensajes contienen cuerpos, en estos casos los paquetes se terminan con un simple CRLF.

En la figura 5.10 se muestra la estructura básica y las diferencias existentes entre los bloques de petición y respuesta.

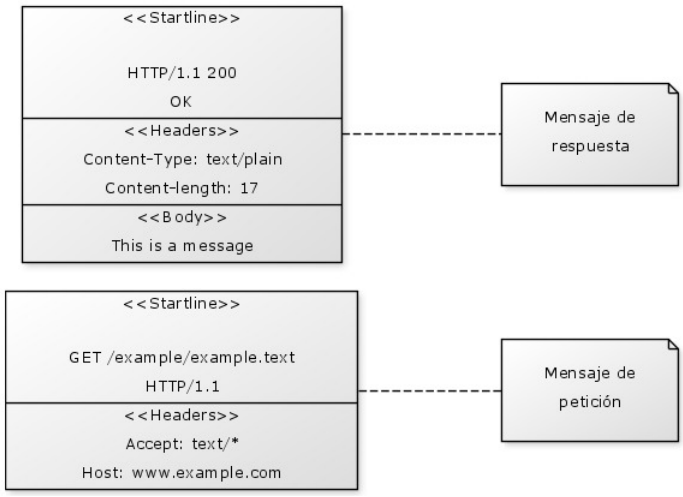

Figura 5.10: Esquema simplificado de los mensajes de respuesta y petición

5.3.5. Lineas de inicio

La linea de inicio de un mensaje de petición dice qué hacer, la linea de inicio de un mensaje de petición dice qué hacer, mientras que la linea de inicio de un mensaje de respuesta dice qué pasó.

1. **Linea de inicio de un mensaje de petición:** Los mensajes de petición le piden a los servidores realizar alguna acción como se mencionó en secciones pasadas. La linea de inicio para estos mensajes contiene la descripción de la operación que se quiere realizar sobre el recurso que describe la URL. Al mismo tiempo incluye la versión del protocolo utilizado que le dice al servidor qué códigos está utilizando el cliente.

2. **Linea de inicio de un mensaje de respuesta:** Los mensajes de respuesta llevan información del estado del servidor así como los datos resultantes de regreso al cliente. Esta linea contiene la versión HTTP del mensaje así como los códigos de estados y los mensajes resultantes de la operación.

3. **Métodos:** Son utilizados para decirle al servidor qué hacer con los recursos que se encuentran dentro del servidor, así como con la información que es enviada.

   No todos los métodos definidos en el estándar HTTP serán utilizados dentro de este proyecto. Gracias a que el estándar fue diseñado para ser modular y extensible, es posible sólo implementar algunos de ellos sin dañar la comunicación entre cliente y servidor.

4. **Códigos de estado:** Así como los métodos le dicen al servidor qué hacer, los códigos de estado le dicen al cliente qué ha sucedido con su petición.

   Cuando un cliente envía une mensaje a un servidor HTTP muchas cosas pueden suceder. Si la petición se hizo correctamente, entonces será exitosa. Pero si la petición no se hizo correctamente, el método no está implementado, no se tienen permisos o el recurso no es encontrado, entonces la petición será fallida.

   Los códigos de estado son retornados para que el cliente sepa cuál ha sido el problema que se ha encontrado.

   Existen diferentes códigos de estado, en la tabla 5.2 se muestran de nuevo las clasificaciones de acuerdo al estándar HTTP/1.1 con los rangos definidos actualmente:

Rango	**Rango definido**	**Categoría**
100-199	100-101	Información
200-299	200-206	Éxito
300-399	300-305	Redirección
400-499	400-415	Error de cliente
500-599	500-505	Error de servidor

Tabla 5.2: Códigos HTTP según su significado

   En la tabla 5.3 se muestra los códigos más comunes utilizados en los servicios web.

Código de estado	**Frase de Razón**	**Significado**
200	OK	El recurso pedido fue exitoso
401	Unauthorized	Es necesarios permisos privilegiados
404	Not found	El recurso no existe en la URL dada
503	Server internal error	Un error ha causado al servidor un fallo crítico

Tabla 5.3: Códigos más utilizados en los servicios web

5. **Frases de razón:** Las frases de razón son el último componente de la linea de inicio de las respuestas. Estos proveen un explicación textual del código de estado.

   Las frases de razón tienen una relación uno a uno con los códigos de estado, solamente

sirven para generar una versión entendible para los humanos de los que ha sucedido con la petición.

La especificación HTTP no provee explícitamente reglas acerca de cuales deben de ser las frases de razón.

6. **Versión:** Las versiones aparecen tanto en las peticiones como en las respuestas dentro de las linea iniciales.

   Son pensadas para proveer a las aplicaciones HTTP con la información necesaria para saber qué operaciones son válidas. Por ejemplo, una aplicación utilizando la versión 1.1 del protocolo comunicándose con otro sistema que utiliza a versión 1.0 debe saber que no debe de utilizar ninguna nueva funcionalidad que haya sido agregada al protocolo.

5.3.6. Cabeceras

Las cabeceras y los métodos son utilizados por el servidor para saber qué hacer con la información. La cabeceras añaden información adicional a los mensajes de petición y respuesta. Son básicamente una lista de pares nombre-valor que permiten transmitir más metadatos.

Existen cabeceras que son específicas para cada tipo de mensaje y cabeceras que tienen un propósito más general, proveyendo información tanto en la petición como en la respuesta.

Las cabeceras pueden ser clasificadas en 5 categorías principales:

1. **Cabeceras generales:** Este tipo de cabeceras genéricas son utilizadas en ambos tipos de mensajes. Son empleadas para muchas soluciones que son útiles a clientes, servidores y otras aplicaciones. Un ejemplo son las cabeceras que manejan la fecha que permiten a ambos lados indicar el tiempo y la hora en la que el mensaje fue creado

   Date: Tue, 10 Sep 2013 09:17:00 GMT

   Algunas cabeceras generales son:

   a) ***Connection:*** Permite a los clientes especificar las opciones acerca de la conexión para los paquetes.

   b) ***Date:*** Provee la fecha y hora en la que el mensaje fue creado.

   c) ***MIME-Version:*** Establece la versión MIME que el transmisor está utilizando.

   d) ***Trailer:*** Lista el conjunto de cabeceras que están en camino para un mensaje codificado que ha sido truncado.

   e) ***Transfer-Encoding:*** Le dice al receptor que código se está utilizado en el mensaje para poder transportarlo de manera segura.

   f) ***Upgrade:*** Establece una nueva versión de algún protocolo que se espera se actualice.

g) ***Via:*** Muestra qué intermediarios ha atravesado el paquete en el camino.

2. **Cabeceras de peticiones:** Como su nombre lo indica, son específicas de los paquetes de petición. Provee información extra a los servidores como el tipo de datos que el cliente quiere recibir. Un ejemplo es la cabecera Accept que le dice al servidor el tipo de dato que se quiere recibir:

   Accept: */*

   Algunas cabeceras de peticiones son:

   a) ***Client-IP:*** Provee la dirección IP de la máquina que contiene el cliente.

   b) ***From:*** Provee el correo electrónico del cliente

   c) ***Host:*** Provee el hostname y el número de puerto hacia donde se debe enviar la respuesta.

   d) ***Refer:*** Provee la URL del documento que contiene la actual URI.

   e) ***UA-Color:*** Provee información acerca de las capacidad de cliente para desplegar color.

   f) ***UA-CPU:*** Provee información general acerca del manufacturero del procesador.

   g) ***UA-Disp:*** Provee información acera de las capacidades del monitor del cliente.

   h) ***UA-OS:*** Provee la información acerca del nombre y la versión del sistema operativo.

   i) ***UA-Pixels:*** Provee la información relacionada con pixeles del monitor.

   j) ***User-Agent:*** Le dice al servidor el nombre de la aplicación que está haciendo la petición.

3. **Cabeceras de respuesta:** Los clientes también necesitan información adicional proveniente del servidor para poder tratar los datos. Estas cabeceras proveen información al cliente acerca de cómo deben de tratarse los datos enviados desde el servidor o información acerca del propio servidor:

   Server: Tiki-Hut/1.0

   Algunas de las cabeceras son:

   a) ***Age:*** Representa qué tan vieja es la respuesta.

   b) ***Public:*** Una lista de los métodos que el servidor soporta para sus recursos.

   c) ***Retry-After:*** Una fecha u hora para intentar de nuevo. Es enviada si el recurso no es accesible por el momento.

   d) ***Server:*** El nombre del servidor seguido de la versión utilizada.

e) ***Title:*** Para los documentos HTML. El título es el que se envía en el documento HTML dentro de las etiquetas <title></title>.

f) ***Warning:*** Un mensaje de aviso más detallado que la frase de razón.

4. **Cabeceras de entidad:** Describen el cuerpo y la información que contiene. Cada instancia establece el tipo de datos que existen en el cuerpo del paquete:

   Content-Type: text/html; charset-utf8

   Algunas de las cabeceras son:

   a) ***Allow:*** Lista los métodos que son permitidos para este paquete

   b) ***Location:*** Le dice al cliente en dónde se encuentra

   c) ***Conetnt-Encoding:*** Informa de cualquier codificación que haya sido hecha en el cuerpo

   d) ***Content-language:*** El lenguaje que se puede utilizar para entender el cuerpo

   e) ***Content-length:*** La longitud del contenido del cuerpo

   f) ***Content-Type:*** El tipo de objeto que se transporta en el cuerpo.

5. **Cabeceras de extensión:** Son cabeceras no manejadas dentro del estándar que pueden ser creadas por desarrolladores y terceros para ser utilizadas dentro de sus paquetes. Todos los sistemas HTTP deben soportar cabeceras de extensión inclusive si no saben cuál es el significado de estas.

5.4. Selección de presentación de recursos

Las API generalmente utilizan un formato basado en texto para representar el estado de los recursos dentro de los servidores. También son utilizados para enviar recursos completos entre computadoras, aunque esta aplicación es más utilizada por servidores web para servir documentos HTML actualmente se utiliza para enviar todo tipo de información a diferentes aplicaciones al mismo tiempo. Para estas nuevas aplicaciones existen dos formatos de representación de datos que son muy comunes JSON y XML. En los últimos años un nuevo formato se ha comenzado a utilizar conocido como YAML.

La representación de datos está pensada para ser un remplazo distinto a los métodos tradicionales que hasta ahora se han utilizado para enviar información entre sistemas como archivos con listas de representación nombre-valor que podemos encontrar en los archivos INI del sistema operativo Windows o los archivos de propiedades que se utilizaban anteriormente en JAVA.

Los sistemas de representación de datos han resuelto el problema de cómo comunicar diferentes sistemas que tienen diferentes necesidades de datos y que se encuentran alejados

geográficamente, ya que permiten que la transferencia de información entre sistemas sea mucho más sencilla y eliminan el problema de tener que preocuparse por diferentes formatos dando a los desarrolladores una mayor abstracción.

Una de las metas de estos formatos es el dividir claramente la información de su representación. Esto significa que la misma información puede tener múltiples representaciones incluso dentro del mismo estándar.

Otro punto en el que este tipo de formatos se distingue sobre simples archivos de texto es en la representación de jerarquías en los datos, por ejemplo la representación de herencia entre objetos en un sistema orientado a objetos o sistema de archivos de una computadora que necesitan un punto raíz de donde comenzará la jerarquía con objetos que pueden tener sub-objetos[6]. Este tipo de anidamiento puede ser infinito dentro de este tipo de formatos, algo que en archivos de texto generaría un gran trabajo.

Finalmente la mayor ventaja que tienen estos formatos es la interoperabilidad. Gracias a que estos formatos se han adaptado como estándares en los últimos años, es mucho más sencillo tomar algunos datos, transformarlos en XML y publicarlos a tener que publicar formatos propios y especificar cómo deben ser procesados. Debido a la gran cantidad de herramientas existentes que permiten la escritura y lecturas de los formatos presentados a continuación, cualquier persona puede publicar y leer en sus aplicaciones la información de otros y enviar su propia información a otros sistemas.

5.4.1. JSON

[44]JSON es un estándar abierto basado en texto que permite el intercambio de información. Es fácil leerlo para los humanos, es independiente de la plataforma en la que se lea y es manejado por una gran cantidad de aplicaciones a nivel global. La codificación no aumenta mucho la cantidad de bytes a la información por lo que es conocido como un formato ligero. Inicialmente fue pensado para ser utilizado con el lenguaje Javascript, aunque su utilización se ha extendido a otros lenguajes que lo utilizan para realizar operaciones, peticiones, respuestas, etc, de manera síncrona o asíncrona.

La sintaxis de JSON fue creado a partir de lo que se conoce como *objects literals* en Javascript, que son representaciones de objetos en los que los atributos y sus valores son escritos y separados por algún símbolo en común.

A pesar de que Javascript tiene una gran flexibilidad con la representación de sus datos, el formato JSON no es tan abierto para mantener compatibilidad entre lenguajes. El estándar establece que el nombre del atributo y el valor deben de estar contenido siempre entre comillas dobles mientras que Javascript permite omitir estas comillas o poner comillas simples en el

[6]La palabra sub-objeto debe entenderse como una analogía a las subclases dentro de la programación orientada a objetos. La referencia de objeto está pensada como un término genérico que represente cualquier tipo de dato, archivo, programa, etc.

nombre del atributo siempre y cuando no sea una palabra reservada.

La ventaja de JSON se encuentra en su simplicidad. Un mensaje con éste formato está compuesto por un objeto en el más alto nivel. Los tipos de elementos guardados dentro de la estructura pueden ser objetos o arreglos. El formato JSON está explicado en su propio estándar RFC 4627 [44].

Uno de los puntos débiles de JSON es que no soporta de manera nativa variables temporales. Esto es debido a que Javascript tampoco soporta este tipo de variables ya que la información temporal es manejada por el objeto Date. Este inconveniente generalmente es arreglado utilizando un objeto aparte que maneje esta información o a través de una variable que lleve información en un formato que pueda ser entendido.

Entre las técnicas más utilizadas se encuentra el poner una representación del tiempo en milisegundos utilizando el sistema POSIX o utilizar una representación más general como "Date(521478544785)".

Las reglas de sintaxis de JSON son las siguientes:

1. Un texto JSON es un sólo objeto o arreglo

2. Un objeto es una lista de miembros dentro de llaves:

$$\{\ \textit{lista de miembros}\ \}$$

3. Un arreglo es una lista de simples valores encerrados entre corchetes

$$[\ \textit{lista de valores}\ ]$$

4. Un miembro en una lista puede ser a su vez otra lista, estar vacío o una pareja nombre valor

$$\text{"}\textit{nombre}\text{"}\ :\ \textit{valor}$$

5. Una lista de valores es simplemente otra lista, un valor vacío o una serie de valores separados por comas

6. Un valor es una cadena, un número, un objeto, un arreglo, un valor booleano, o estar vacío. Para las cadenas es necesario utilizar secuencias de escape para representar elementos como las comillas o el guión medio.

5.4.2. XML

XML proviene de *Extensible Markup Language* y el desarrollo de este lenguaje ha experimentado lo que se podría considerar como un camino común dentro de las tecnologías de la información y la comunicación. Comenzó en los años 90's como una solución a un problema particular que muy pocos utilizaban, después se fue extendiendo su uso conforme las computadoras aumentaban su

capacidad de procesamiento y las herramientas para la traducción de XML fueron mejorando, eso generó que el lenguaje se volviese mucho más sofisticado y generalizado. Últimamente ha existido una disminución en el uso de este lenguaje debido a una cantidad de problemas y puntos débiles que tienen las herramientas de traducción y el nacimiento de nuevas alternativas, sin embargo sigue siendo un lenguaje ampliamente utilizado en sistemas de configuración y algunas API alrededor del mundo.

Este formato es en realidad una versión simplificada de SGML[7], que es un estándar de documentación internacional desde los 80's. Sin embargo, SGML es demasiado complejo especialmente para la web. Por ello Jon Bosak desarrolló un grupo en la W3C que permitiera crear un nuevo lenguaje derivado de este para su utilización en el internet.

XML puede ser visto como un metalenguaje que permite crear y expandir formatos propios. Esto lo diferencia de lenguajes como HTML en donde ya existen etiquetas establecidas, por ello no es posible crear etiquetas propias y esperar que exista una comunicación correcta con el receptor. En XML es permitido crear etiquetas con nombres propios y configurar las opciones de estas de manera propia.

Para mantener la separación de los datos y su representación, XML define reglas que son obligatorias y que permite uniformidad entre sistemas:

1. Un documento XML consiste de uno o más elementos. Estos elementos comparten un formato en común

   <identificador elemento><— etiqueta de apertura
   elemento
   </identificador elemento><— etiqueta de cierre

2. Las etiquetas deben encontrarse entre pico paréntesis (<>) y las etiquetas de cierre deben de contener una diagonal que las identifica (/)

3. Las etiquetas pueden tener atributos en ellas que agreguen información acerca del elemento que encierran. Los valores de los atributos deben de estar siempre entre comillas. Un ejemplo puede ser

   <auto fabricado="México»
   Volks Wagen sedan
   </auto>

4. En las etiquetas se hace distinción entre mayúsculas y minúsculas, por lo que <teléfono> y <Teléfono> son etiquetas distintas.[8]

[7] SGML: Standard Generalized Markup Language

[8] Esta es otra de las distinciones entre XML y HTML en donde el segundo no discrimina entre mayúsculas y minúsculas

5. Todo elemento no nulo debe de tener siempre una etiqueta de apertura y una de cierre. Si esto no se cumple la información se perdería en los traductores.

<auto fabricado="México»
Volks Wagen sedan
</auto>
Tsuru <— **incorrecto**

6. Las etiquetas deben de respetar la jerarquía con la que fueron creadas. Un documento XML bien formado se dice que cumple con esta regla en todos sus elementos. Un ejemplo de un error común puede ser

<italic><bold>Texto</italic></bold>

7. Los archivos de XML siempre deben de contener la definición del tipo de documento que es utilizada para saber qué versión de XML se utilizará, que codificación se usa para la información[9], entre otras cosas.

<?xml version="1.0.^encoding=ÜTF-8¿><!DOCTYPE Tesis SYSTEM "sample.dtd»

Como se mencionó anteriormente, actualmente han surgido alternativas al lenguaje XML que intentan resolver algunos problemas que se han encontrado con los traductores. El primer problema lo encontramos en que la codificación debe ser especificada, lo que implica que si el servidor está utilizando algún tipo de codificación que no se encuentre dentro del cliente este no podrá leer la información.

El segundo problema que se encuentra es en el tamaño que los archivos XML puede alcanzar para conjuntos de datos muy grandes. Esto se debe principalmente a las etiquetas, las cuales añaden una gran cantidad de caracteres que no se encuentran en la información original.

El tercer problema que tiene XML es la velocidad que pueden llegar a tener los traductores. La manipulación del DOM[10] se realiza a través de modelos de grafos que mapean la estructura y después buscar a través de ella la información. Esto genera que su traducción sea mucho más lenta que la traducción de modelos como JSON o YAML en los que se puede mapear directamente entre la estructura y los objetos a crear.

5.4.3. YAML

YAML proviene del acrónimo *YAML Ain't Markup Language*. Es un sistema de representación de datos que fue diseñado con la principal intención de ser amigable a la lectura humana y trabajar bien con los lenguajes de programación actuales.

[9] A diferencia de JSON en donde el estándar establece que siempre se debe usar la codificación utf-8, en XML es necesario especificar esto.

[10] DOM: Document Object Model

Las herramientas abiertas que permiten interoperabilidad y que son fácilmente leíbles por los humanos han permitido el desarrollo de Internet. YAML fue diseñado desde el inicio para ser útil y amigable con las personas que necesitan trabajar con datos. Utiliza caracteres basados en el estándar Unicode algunos proveyendo información estructural y otros los datos.

Una de las ventajas de YAML es la sencillez de estructura que es necesaria para poder expresar los datos, las reglas de sintaxis de este estándar son

1. La indexación es utilizada para presentar estructura y jerarquía. De esta manera cada nivel en la indexación representa un nivel más profundo dentro del objeto

```
Campo 1 (nivel 1):
                    Subcampo 1 (nivel 2):
                    Subcampo 2 (nivel 2):
                    Subcampo 3 (nivel 2):
                    Subcampo 4 (nivel 2):
                                        Subsubcampo 1 (nivel 3):
Campo 2 (nivel 1):
                    Subcampo 1 (nivel 2):
                                ...
```

2. Dos puntos son utilizado para separar el nombre del atributo de su valor

$$nombre \ : \ valor$$

3. Las lineas diagonales son utilizadas para crear colecciones o listas

YAML intenta diseñar una interfaz abstracta liberándose de las estructuras de datos que sea definidas por implementación dentro de un lenguaje. Esto lo hace utilizando un mecanismo de tipos de datos basados en cadenas que se puede traducir a cualquier tipo de estructura de datos.

La indexación de YAML es parecida a la que presenta el lenguaje Python sin la ambigüedad de los tabuladores. Los bloques de indexación facilitan la inspección de la información para los humanos además y facilitan la lectura de estos textos en lenguajes como Perl, Python y Ruby. Este tipo de estándar soporta colecciones como mapas, secuencias y escalares.

5.4.4. Comparación entre las tecnologías de representación de datos

Tanto XML, JSON como YAML pueden ser utilizados para representar objetos en memoria dentro de un formato de texto que es entendible para un humano y permite el intercambio de información. Los tres formatos son considerados isomorfos[11], sin embargo eso no implica que sea necesario implementar todos.

[11] Isomorfismo: Propiedad de las representaciones de datos que establecen que dada una representación de información en uno de estos formatos, es posible crear otro formato que contenga lo mismo en el mismo orden.

Esto implica que escoger entre cuál formato se debe de utilizar es necesario estudiar las características de la necesidad que se esté resolviendo y en particular de la aplicación. Por ejemplo el formato XML puede ser principalmente utilizado en aplicaciones que necesiten documentos con etiquetas de marcado (como procesadores de HTML) y en donde la cantidad de información no es enviada. JSON por otro lado, es principalmente utilizado en aplicaciones que implican lenguajes de programación que pueden mapear más rápidamente el formato hacia sus tipos de variables. Finalmente YAML está siendo utilizado actualmente es aplicaciones que requiere sistemas de configuración jerárquicos sin necesidad de escribir un documento XML completo.

Es por ello que en el tabla 5.4 se comparan los tres lenguajes:

Características	**JSON**	**XML**	**YAML**
Permite expresar conjuntos de datos (arreglos)	X		X
Permite identificar tipos de datos	X	X	X
Permite expresar objetos	X		X
Permite soporte de datos nulos	X	X	X
Indexación obligatoria			X
Permite el soporte de comentarios		X	X
Nombres de dominio (namespaces)		X	
La información relevante ocupa la mayor parte del paquete	X		
Fácilmente traducido entre diferentes lenguajes	X		
Depende de la plataforma en la que se desarrolle			X
Tiene una curva de aprendizaje suave	X		X
Gran soporte entre plataformas y lenguajes	X	X	

Tabla 5.4: Comparación entre JSON, XML y YAML

A pesar de que JSON es relativamente nuevo, ha sido su sencillez y versatilidad lo que le ha permitido ser uno de los formatos más soportados hoy en día. Es por ello que se eligió para el desarrollo de este proyecto como formato por defecto; sin embargo, no por ello se debe entender que el formato XML ha quedado descartado, ya que existirán clientes que lo requerirán posiblemente en un futuro por lo que siempre se recomienda que se le de soporte.

En el caso del formato YAML tiene dos problemas que por ahora se deben tomar en cuenta, la dificultad que se presenta el transformar un tipo de dato en este formato debido a las indexaciones y el poco soporte que actualmente existe dentro de las comunidades lo que genera que no existan muchas herramientas para mapeo, es por ello que se descarta como una solución por el momento para poder ser el formato de intercambio. Sin embargo es un formato que puede tener una gran cantidad de aplicaciones en un futuro por lo que es importante tomarlo en cuenta.

5.4.5. Protocol Buffer

A diferencia de todos los modelos de representación vistos hasta ahora, éste es obligatorio debido a que los sistemas en tiempo real de GTFS exigen que la transferencia de información se realice en este tipo de formato.

Los protocol buffers tienen la característica de ser representaciones binarias, lo que implica que está optimizado para el entendimiento entre máquinas y no para humanos. Esto permite una estructura mucho más pequeña y optimizada que da como resultado una menor cantidad de bits enviados y procesados aumentando la cantidad de clientes con la misma infraestructura.

En la página principal para este protocolo [21] se establece que son un mecanismo de representación neutro en términos de lenguajes y plataformas, además de extensibles. Para este protocolo se crean archivos que permiten establecer cómo será estructurada de la información y después son utilizados para saber cómo serán los mensajes de salida en el cliente y en el servidor.

La manera en cómo trabajan este tipo de formatos es a través de un archivo cuya extensión es ".proto", este archivo es el que contiene las estructuras de la información que se va a mandar basado en una reglas simples:

1. Los archivos deben de comenzar con la palabra message seguida del nombre de la clase que se creará para la interfaz
   message NombreClase {
   attributos a enviar
   }

2. Existen tres operadores modificadores para los atributos que pueden ser utilizados:

   a) **required:** Son atributos que deben de poseer siempre un valor, si no es así, el paquete es considerado como inválido

   b) **optional:** Son atributos que pueden no estar o no tener un valor dado. En ese caso es posible obtener un valor por default cuando no es enviado.

   c) **repeated:** Representa los atributos que trabajan como colecciones. Estos atributos puede ser repetido tantas veces como se necesite.

3. Los atributos deben de especificar el tipo que van a contener y la posición dentro del paquete que va a ocupar, de esa manera se puede optimizar el paquete aún más.

   $$\text{int32 atributo} = 1;$$

   Los tipos de datos que se permiten son:

   a) **double**

   b) **float**

c) **int32:** Utiliza codificación por longitud de variable. Es ineficiente para enteros con signo.

d) **int64:** Utiliza codificación por longitud de variable. Es ineficiente para enteros con signo.

e) **uint32:** Utiliza codificación por longitud de variable.

f) **uint64:** Utiliza codificación por longitud de variable.

g) **sint32:** Utiliza codificación por longitud de variable. Este tipo de variable es eficiente para números con signo.

h) **sint64:** utiliza codificación por longitud de variable. Este tipo de dato es eficiente para números con signo.

i) **fixed32:** Tiene una longitud fija de 4 bytes. Es más eficiente que unit32 si lo números son mayores que 2^{28}.

j) **fixed64:** Tiene una longitud fija de 8 bytes. Es muy eficiente si se intenta representar números mayores a 2^{56}.

k) **sfixed32:** Tiene una longitud fija de 4 bytes.

l) **sfixed64:** Tiene una longitud fija de 8 bytes.

m) **bool**

n) **string:** Una cadena de contener caracteres utf-8 o ascii

ñ) **bytes:** Puede contener cualquier secuencia de bytes.

4. Cada uno de los campos debe tener una etiqueta numérica que lo represente. Estas etiquetas son utilizadas dentro de mensaje en binario para saber la posición dentro del archivo que ocupa el atributo, por lo que no deberían ser modificadas durante la comunicación. Los número del 1 al 15 pueden ser codificados con sólo un byte por lo que se recomienda poner en estos rangos las etiquetas required o los atributos que más se accedan.

```
message NombreClase {
required string atributo = 1;
optional int32 atributo_2 = 2;
repeated int32 atributo_3 = 3;
}
```

Una vez que se ha finalizado el archivo se pasa a través de un intérprete que genera un archivo con meta clases las cuales son utilizadas para crear los paquetes que se enviarán. Una de las ventajas de este protocolo es que el compilador genera código dependiendo del lenguaje para el que fue diseñado.

El estándar GTFS en tiempo real provee un archivo proto que contiene toda la información necesaria para crear las clases que deben ser utilizadas cuando la información es presentada al público. De esta manera se aseguran que todos cumplen con el mismo formato y atributos.

5.5. Diseño de los metadatos

Existen una gran cantidad de metadatos que pueden ser enviados a través de las cabeceras HTTP tanto en las peticiones como en las respuestas. El protocolo define una serie de cabeceras estándar que proveen información acerca de el recurso al que se quiere acceder, mientras que otras cabeceras dan detalles acerca de la información que el paquete lleva consigo. Finalmente se tienen las cabeceras que funcionan como directivas de control.

En esta sección se da una propuesta de las reglas que deben de seguir las cabeceras que se utilizaron en este proyecto de tesis para que se sigan los estándares más utilizados en la industria.

1. **La cabecera Content-Type debe ser siempre utilizada:** La cabecera *Content-Type* provee el tipo de dato que se encuentra dentro del cuerpo del paquete. El valor de esta cabecera es un formato de texto especial conocido como *media type*. Los servidores y clientes aprovechan este campo para saber cómo procesar la información del cuerpo del mensaje.

2. **La cabecera Content-Length debe ser utilizada:** Esta cabecera describe la longitud del cuerpo del mensaje en bytes, lo cual hace a esta cabecera muy importante en la comunicación ya que permite saber si el paquete está completo cuando llega. También permite saber la longitud del paquete a través del verbo HTTP HEAD sin necesidad de descargar el paquete.

3. **La cabecera Last-Modified debe ser utilizada en los paquetes de respuesta:** Esta cabecera puede ser utilizada para aumentar la velocidad de respuesta y disminuir la sobrecarga del servidor al permitir saber si la información o el recurso que se está pidiendo ha sido modificado desde la última petición. Este campo guarda una cadena de texto que representa el tiempo de la última vez que el fue modificado.

   Esta cabecera debe ser siempre incluida en paquete que respondan a los métodos GET y HEAD.

4. **La cabecera Etag es recomendable que sea incluida:** Aunque esta cabecera parece algo complicada cuando se analiza el protocolo HTTP [43], es una cabecera muy útil cuando se intenta reducir el ancho de banda y cantidad de carga en el servidor. Esta bandera contiene una cadena que es un identificador especial para una versión dada de un recurso, cada vez que el recurso es actualizado el valor de este campo es modificado. Esto puede ser utilizado por sistemas de cache y servidores que no tienen que responder con el recurso siempre.

   Aunque se podría pensar que esta cabecera tiene una funcionalidad idéntica a *last-modified*, en realidad se diferencian en que son los clientes los que a través del campo

If-None-Match envían esta cadena para que el servidor compare la versión que tienen en sus máquina con la actual versión.

Esta cabecera es especialmente importante en redes de deliberación de contenido, en donde cada cierto tiempo los nodos CDN realizan una petición de actualización al servidor central y modifican su fecha de modificación del recurso aún cuando el contenido no ha sido modificado. Esto haría que con la cabecera HTTP *last-modified* pareciese que existe un nuevo recurso, algo que se evita con la cabecera *Etag*

5. **Es recomendado que los sistemas de almacenado soporten peticiones condicionales:** Esto es debido a que puede permitir resolver muchos potenciales problemas por ambigüedad al recibir un paquete hacia un recurso no existente generando que el servidor no sepa si se intenta crear el recurso o sólo actualizar el recurso si es que existiese.

   Para realizar este tipo de tipo de razonamiento se utilizan las cabeceras *If-Unmodified-Since* e *If-Match* que permiten saber al servidor si aceptar la petición hacia un recurso si éste no ha tenido una modificación en un tiempo dado o que cumplan con una cadena de identificación respectivamente.

   La primera cabecera en una petición le dice a la API que acepte el recurso sí y sólo sí no ha existido un cambio en su representación en un tiempo, mientras que la segunda permite que la petición sea condicional al basarse en un identificador que es creado a través recurso en su estado actual. Ambas cabeceras permiten consultar si es necesario obtener un recurso nuevo antes de que este sea descargado lo que ahorra mucho ancho de banda y reduce la carga en el servidor.

   Estas cabeceras también se pueden utilizar para reducir el conflicto de ambigüedad que se mencionó al inicio al permitir a los clientes preguntar por el estado de un recurso antes de querer modificarlo al mandar la cabecera *If-Match* y establecer si el objeto ha sido modificado o creado antes de que nuestra petición llegase al servidor.

6. **La cabecera Location puede ser utilizada para responder la URI de un nuevo recurso creado:** Si el usuario necesita saber cuál es la URI final de un recurso que acaba de crear después de que fuese exitosa la operación se puede enviar esta cabecera junto con el paquete de notificación de éxito.

7. **Las cabeceras de Cache-Control, Expires y Date deben de ser utilizadas si el sistema usa cache:** Los sistemas intermedios que entregan información desde una caché al usuario son hoy en día una de las normas de la construcción de servicios web. La razón es que permiten atender a más usuarios al mismo tiempo y reducen el tiempo de respuesta al poder poner sistemas muy cercanos geográficamente.

   Las ventajas de los sistemas de cache es que pueden estar en cualquier sitio, desde el servidor hasta sistemas de deliberación de contenido o inclusive en el propio dispositivo

final del usuario.

Para que se cumpla el estándar HTTP/1.1, también se debe de incluir la cabecera *Expires* la cual contiene un tiempo de expiración. Este valor representa el tiempo en el que la API generó el recurso más un tiempo de vida que tendrá el recurso antes de que deje de ser válido. También se puede incluir la cabecera Dates que permite mandar la fecha en la que el recurso fue enviado por la API.

8. **Si no se utilizan sistemas de caché entonces deben de usarse las cabeceras Cache-Control, Expires y Pragma:** En este proyecto no se utilizó ningún sistema de caché. Cuando esto sucede, el protocolo HTTP/1.1 establece que deben de usarse las cabeceras *Cache-Control* con el valor **no-cache** y **no-store**, al igual que las cabeceras *Pragma* con el valor **no-cache** y la cabecera *Expires* con el valor 0.

9. **No se deben de utilizar cabeceras personales para modificar el comportamiento de los métodos HTTP:** El protocolo HTTP permite agregar cabeceras personales a los paquete, aunque en el protocolo se establece claramente que deben de ser utilizados solamente para propósitos de añadir información. [43]

   Si la información que se envía a través de estas cabeceras es utilizada para modificar la manera en cómo se interpreta la información o ayuda en su interpretación, entonces está información debe ser mandada a través del cuerpo del paquete.

5.5.0.1. Media Types

Este tipo de cabeceras sirve para describir el formato en el que la información se encuentra. Esta información se envía a través de la cabecera *Contet-Type*.

El formato del valor de esta cabecera es:

tipo ”/“ subtipo (”;“ parámetros)

El tipo puede ser: **application, audio, message, image, model, multipart, text o video**. Para la API de este proyecto se utiliza **application**.

La IANA[12] maneja los *media types*[13] que se encuentran actualmente estandarizados y los provee en diferentes RFC[14] [47]. Algunas combinaciones comúnmente utilizadas de estas cabeceras son:

1. **text/plain:** Representa un paquete que contiene texto sin formato específico.

2. **text/html:** Representa un paquete que contiene texto con formato HTML[15].

[12]Internet Assigned Numbers Authority

[13]Media types fueron originalmente conocidos como Myme types que era el acrónimo de Multiporpose Internet Mail Extensions

[14]RFC: Request for comments

[15]HTML: HyperText Markup Language

3. **image/jpeg:** Las imágenes comúnmente utilizan el subtipo para establecer qué método de compresión fue utilizado para crearla.

4. **application/xml:** El paquete contiene información estructurada utilizando el formato XML [45]. Esta cabecera es la que debe de enviar el cliente dentro de nuestro servidor cuando se quiera que la respuesta sea utilizando este formato.

5. **application/atom+xml:** Este tipo de esquema es más utilizado con aplicaciones que envían pequeñas porciones de datos utilizando el formato Atom[16] el cual está basado en xml.

6. **application/javascript:** Representa un paquete que contiene código javascript en él.

7. **application/json:** El paquete contiene texto en formato JSON. [44] Este formato es el que se utiliza por defecto en la API si no se especifica esta cabecera.

Existen algunos puntos comunes que se utilizan en los servicios web cuando se trabaja con este tipo de cabeceras, sin embargo no existen reglas estrictas de cómo deben ser utilizadas ya que cada desarrollador puede soportar de distinta manera la misma información.

Las reglas más comunes en los sistemas de servicios web son:

1. **El servidor debe soportar negociaciones de media types cuando múltiples representaciones de la misma información estén disponibles:** Permitir a los clientes la negociación del tipo de formato que también puede ser aceptado en caso de que no se encuentre el que se solicita a través de la cabecera *Content-Type* ayuda a facilitar el trabajo al cliente y a los desarrolladores. Esto se logra enviando el *media type* deseado a través de la cabecera *Accept*.

   Content-Type: application/json
   Accept: application/xml

   Esta cabecera permite pedir información en formato JSON y XML si el primero no se puede enviar.

2. **El servidor puede permitir escoger el media type a través de la URL:** Utilizando atributos debe ser posible escoger en tipo de formato alternativo que se quiere emulando el trabajo de la cabecera *Accept*

   GET http://.../vehicles/254?accept=application/xml

   Debe dar un resultado similar al ejemplo que se puso en el punto anterior. Sin embargo este tipo de petición pone la información sobre la URI lo que lo hace propenso a vulnerabilidades de seguridad y por lo que sólo se debe utilizar como último recurso.

[16] Atom: Atom Syndication Format

5.6. Vistas de la respuesta de la API

La API tiene dos formas de interactuar con el mundo externo. La primera es introduciendo información del exterior a la base de datos utilizando las herramientas desarrolladas. En la figura 5.11 se puede ver una captura de pantalla de la API introduciendo actualizaciones de tránsito para el feed Transit Updates. En las figuras 5.12 y 5.13 se puede ver esta cómo se ven las bases de datos mientras se llenan.

La segunda forma de interacción es enviando información hacia el mundo externo. Son embargo esto se hace a través de fuentes binarias por lo que no es legible para el humano como se ve en la figura 5.14 que muestra la respuesta de la API y 5.15 que muestra esa misma salida en en un navegador.

```
(tesis_env)otero@otero:~/Documents/Codes/tesis_env/tesis/gtfs_real_time$
 python gtfsrdb.py -t http://api.bart.gov/gtfsrt/tripupdate.aspx -a http
://api.bart.gov/gtfsrt/alerts.aspx -d postgresql://otero:
@127.0.0.1/tesis_development
Advertencia: No se incluyó la URL para la descarga de la posición de los
 vehículos
Añadiendo 75 actualizaciones de viajes
Añadiendo 77 actualizaciones de viajes
Añadiendo 76 actualizaciones de viajes
Añadiendo 76 actualizaciones de viajes
Añadiendo 76 actualizaciones de viajes
```

Figura 5.11: Aplicación introduciendo información a la base de datos

```
  route_id    | trip_id |        trip_headsign        | schedule_relationship | id | agency_name
--------------+---------+-----------------------------+-----------------------+----+-------------
 ROUTE_18226 | 38834   | Indios Verdes - El Caminero | SCHEDULED             |  1 | Metrobús
 ROUTE_18226 | 38834   | Indios Verdes - El Caminero | SCHEDULED             |  1 | Metrobús
 ROUTE_18226 | 38834   | Indios Verdes - El Caminero | SCHEDULED             |  1 | Metrobús
 ROUTE_18226 | 38834   | Indios Verdes - El Caminero | SCHEDULED             |  1 | Metrobús
 ROUTE_18226 | 38834   | Indios Verdes - El Caminero | SCHEDULED             |  1 | Metrobús
 ROUTE_18226 | 38834   | Indios Verdes - El Caminero | SCHEDULED             |  1 | Metrobús
 ROUTE_18226 | 38834   | Indios Verdes - El Caminero | SCHEDULED             |  1 | Metrobús
 ROUTE_18226 | 38834   | Indios Verdes - El Caminero | SCHEDULED             |  1 | Metrobús
 ROUTE_18226 | 38834   | Indios Verdes - El Caminero | SCHEDULED             |  1 | Metrobús
 ROUTE_18226 | 38834   | Indios Verdes - El Caminero | SCHEDULED             |  1 | Metrobús
(10 rows)

~
```

Figura 5.12: Vista de una base de datos para Trip Update

route_id	trip_id	trip_headsign	schedule_relationship	stop_id	arrival_delay	id	agency_name
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	HAYW	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	LAKE	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	BALB	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	POWL	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	GLEN	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	DALY	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	WOAK	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	DALY	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	BAYF	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	FRMT	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	FRMT	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	SHAY	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	FRMT	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	LAKE	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	FTVL	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	NBRK	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	RICH	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	RICH	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	LAKE	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	HAYW	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	EMBR	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	ROCK	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	CONC	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	NCON	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	PITT	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	PITT	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	BALB	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	SFIA	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	SFIA	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	MLBR	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	BALB	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	MLBR	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	MCAR	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	CIVC	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	DALY	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	DALY	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	COLS	0	1	Metrobús
ROUTE_18226	38834	Indios Verdes - El Caminero	SCHEDULED	CAST	0	1	Metrobús

:_

Figura 5.13: Vista de la unión de algunas bases de datos

5.7. Proposiciones para la API

5.7.1. Seguridad en la API

Crear una API segura es una decisión crítica que debe ser estudiada desde las primeras versiones al igual que se estudian otras operaciones. En esta sección no se hablará de técnicas de seguridad en internet, solamente se subrayarán aquellas situaciones que afecten la operación directa de la API.

Los modelos de seguridad que se escojan para la API deben de estar de acorde con la importancia de la información que es manejada. Por ejemplo, en una API que maneja información personal de los usuarios la seguridad debe ser muy alta, mientras que en la API utilizada para este tipo de proyecto la seguridad puede ser mucho más holgada.

Los puntos que se deben de analizar cuando se pone la seguridad de la API en funcionamiento son:

1. Qué tipo de información debe enviarse de manera segura
2. Cuánta seguridad es necesaria para cada tipo de información
3. Cómo se va a medir el impacto de la seguridad en la velocidad de reacción del sistema

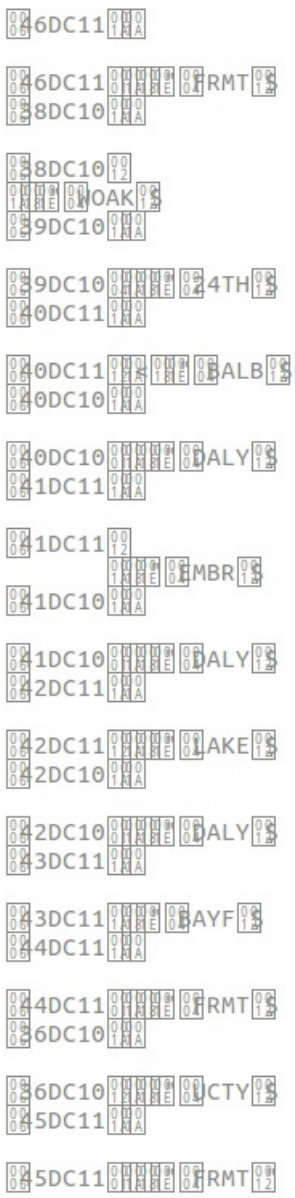

Figura 5.14: Vista de la respuesta binaria de la API

1.0®Ã¦“$ 37DC10 37DC10"LAKE$ 38DC10 38DC10"16TH$ 39DC10 39DC10"DALY4 40DC11* 40DC11<"MONT "POWL$ 40DC10 40DC10"DALY$ 41DC11 41DC11 "LAKE$ 41DC10 41DC10"DALY$ 42DC11 42DC11"BAYF$ 43DC11 43DC11"UCTY$ 35DC10 35DC10"UCTY$ 44DC11 44DC11"FRMT$ 36DC10 36DC10<"SANL$ 45DC11 45DC11" FRMT" 47R11 47R11"SHAY" 48R11 48R11"FRMT" 43R10 43R10"SHAY" 49R11 49R11"FRMT" 44R10 44R10"COLS" 50R11 50R11"FRMT" 45R10 45R10<"MCAR" 46R10 46R10"DELN" 47R10 47R10"RICH" 44R11 44R11<"PLZA" 48R10 48R10"RICH" 45R11 45R11"MCAR" 49R10 49R10"RICH" 46R11 46R11"COLS& 53SFO11 53SFO11"DALY& 55SFO11 55SFO11"DALY& 51SFO11 51SFO11"DALY& 52SFO11 52SFO11"SFIA& 54SFO11 54SFO11"SFIA& 56SFO10 56SFO10<"BALB& 56SFO11 56SFO11"SFIA& 57SFO10 57SFO10 "EMBR6 58SFO10+ 58SFO10 <"MCAR "19TH& 59SFO10 59SFO10"LAFY6 60SFO10+ 60SFO10<"NCON "EMBR& 61SFO10 61SFO10"PITT6 47SFO11+ 47SFO11x"CONC "NCON& 62SFO10 62SFO10"PITT& 48SFO11 48SFO11"LAFY& 63SFO10 63SFO10"PITT& 49SFO11 49SFO11<"19TH& 50SFO11 50SFO11<"POWL& 44DCM10 44DCM10"16TH & 45DCM11 45DCM11"MLBR& 45DCM10 45DCM10 "WOAK& 46DCM10 46DCM10"DBRK& 47DCM10 47DCM10"RICH& 48DCM10 48DCM10"RICH& 39DCM11 39DCM11" DBRK& 49DCM10 49DCM10"RICH& 40DCM11 40DCM11"WOAK& 41DCM11 41DCM11"24TH& 42DCM11 42DCM11"SSAN& 43DCM11 43DCM11"MLBR& 43DCM10 43DCM10"COLM& 44DCM11 44DCM11"MLBR& 44DCM20 44DCM20 "EMBR& 49DCM21 49DCM21"DALY6 45DCM20+ 45DCM20<"COLS "EMBR& 46DCM20 46DCM20"CAST & 47DCM20 47DCM20"DUBL& 48DCM20 48DCM20"DUBL& 44DCM21 44DCM21"CAST& 49DCM20 49DCM20"DUBL& 45DCM21 45DCM21"FTVL& 46DCM21 46DCM21 <" EMBR& 47DCM21 47DCM21"BALB& 48DCM21 48DCM21"DALY

Figura 5.15: Vista de la respuesta binaria de la API en un navegador

4. Cómo se va a medir si las medidas están funcionando

5. Qué tanto se complica el cada submódulo cuando se le agregan las políticas de seguridad

6. Es necesario tener sistemas de autenticación de alta seguridad (Generalmente utilizados para sistemas con tarjetas de crédito)

7. Es necesario identificar otra información del usuario además de su nombre de usuario

Existen 3 niveles de autenticación que se pueden utilizar en una aplicación dependiendo de la cantidad de seguridad que estos necesiten. Los niveles son:

1. **Identificación:** Uno de los primeros pasos que se debe de asegurar es que la API debe saber en todo momento quién le envía información. Un método muy común, que implica un nivel muy bajo de complejidad, es determinar quién está utilizando la aplicación en un momento dado a través de identificadores únicos.

   Los identificadores permiten utilizar las aplicaciones sin las complejidades de las contraseñas y las autenticaciones. Consisten de cadenas aleatorias que los desarrolladores generalmente pasan a través de las cabeceras HTTP.

   Para poder crear una aplicación para la API es necesario que se registre y se obtenga una llave de identificación. Ésta permite mantener un control de quién está enviando esta información ayudando a tener datos estadísticos del uso de la API por los usuarios.

   Las cadenas de texto que se envían son deliberadamente simples, ya que no se requiere un nivel de seguridad muy alto, lo que genera que un atacante pueda dar con una clave y generar falsificación de datos o incluso dañar los módulos internos.

   Un nivel más alto de seguridad es a través de nombres de usuario y contraseñas.

2. **Autenticación:** Este nivel de seguridad resuelve el problema que tienen los problemas de identificación en cuanto a que no es posible asegurar si es la misma persona la que se conecta con una llave. Este tipo de métodos incluyen nombres de usuario, contraseñas y estándares de autenticación como SAML, X.509 o OAuth 2.

   a) ***Nombre de usuario y contraseña:*** Comúnmente las llaves de las API se envían sin cifrar a través de la red. Aunque esto puede reducir la cantidad de procesamiento que se realiza en el servidor, al contrario de lo que podría pensarse, esto no necesariamente permite manejar mayor cantidad de usuarios ya que la comunicación es asíncrona en la mayoría de los casos lo que permitiendo procesar las peticiones conforme los recursos en el servidor se vayan liberando.

   La autenticación por nombre de usuario y contraseña no permite, tan fácilmente, ataques de falsificación. La manera más sencilla de autenticación es a través del protocolo *HTTP Basic authentication* utilizado en una gran cantidad de páginas web y que es soportado por casi todos los servidores de manera nativa. Este protocolo no exige ningún tipo de procesamiento como cifrado o envío a través de redes privadas, por lo que el usuario debe de asegurarse que la información viaje a través de un canal seguro para mantener la contraseña segura.

Este protocolo también trabaja bien para comunicación entre aplicaciones y sólo es necesario asegurarse que la contraseña utilizada en la comunicación sea lo suficientemente fuerte y se encuentre cifrada para que el sistema sea seguro.

b) ***Autenticación basada en sesiones:*** Aunque esta tecnología ya no es usada en casi ningún lugar, la razón por la que se menciona en este trabajo es para darla a conocer como una alternativa existente, además de que es mencionada en muchos libros de servicios web de antes de 2008 [8]. En este sistema el usuario manda a llamar un login que es desplegado para enviar el nombre y contraseña a través del método POST, luego es enviada una respuesta por parte del servidor con una llave de sesión. El usuario debe enviar esta llave en cada una de las peticiones y llamar "logout" cuando ya no requiera más esa sesión.

En algunos escenarios este tipo de autenticación aún funciona, sin embargo, los métodos basados en cookies o llaves de sesión no son tan robustos debido a que enfrentan a problemas similares a los de identificación mencionados anteriormente, al basarse en una clave de texto que puede ser extraída por un atacante.

c) ***OAuth 2:*** Oauth 2 es un protocolo abierto que permite autorizaciones para API desde cualquier dispositivo a través de un método estándar simple. Este maneja las negociaciones entre las aplicaciones y es usado cuando un diseñador quiere saber quién se está conectando a su sistema. Es por ello que es actualmente uno de los protocolos más extendidos de autenticación en servicios web, plataformas en la nube y servidores web.

En lugar de tener una contraseña que funciones como llave maestra a través de todas las aplicaciones, Oauth 2 crea un token, similar a una llave de identificación, que se utiliza para acceder a la aplicación. La diferencia con los métodos basados en sesiones o identificación es que se tiene una clave para cada usuario por cada dispositivo. Esto implica que un atacante debe de realizar la acción desde el dispositivo del usuario para poder tener éxito. Pero también permite que si el dispositivo es robado, el atacante no pueda obtener la contraseña del usuario ya que toda la comunicación se hace a través de claves.

Este protocolo también soporta cifrado nativo a través una palabra conocida como secreto[17], la cual se envía junto con la llave para que sólo el cliente pueda descifrar el mensaje.

El protocolo Oauth 2 es demasiado extenso para ser cubierto por un sólo trabajo. Sin embargo, es un protocolo muy bien documentado [51].

3. **Autorización:** Es el nivel más restrictivo ya que permite el acceso sólo a zonas de la aplicación en las que se tiene permiso. Este tipo de seguridad es utilizada principalmente

[17] Palabra utilizada como semilla en el algoritmo de cifrado

para APIs que soportan una gran cantidad de aplicaciones distintas y de usuarios distintos.

A pesar de que es el nivel más avanzado de todos en seguridad ya que implica un análisis del sistema a niveles muy profundos, también es el único nivel que se desarrolla completamente dentro del servidor por lo que cada uno de los módulos debe contener un submódulo que establezca si la persona que se está autenticando tiene permiso de acceder o un módulo central como el establecido en POL_DOC_3.

Generalmente los tres niveles son utilizados para la autenticación e identificación del usuario. Sin embargo, cada nivel superior implica el desarrollo de nuevos sistemas y hace más compleja la aplicación. Es por ello que se debe analizar si son necesarios los tres niveles para nuestra aplicación en particular.

Algunas API sólo necesitan establecer la identidad del emisor sin necesidad de saber manejar permisos ya que no se alteran recursos en el servidor, mientas que otras necesitan autorización en cada uno de los submódulos para poder utilizarla.

Un buen manejo de usuarios puede generar una enorme reducción en los costos al manejar menor cantidad de información y eso aumenta la satisfacción de los clientes. Esto puede incluir el permitir a los desarrolladores el inscribirse en la plataforma para adquirir una llave de autenticación.

Es posible reducir la cantidad de recursos que se necesitan al crear múltiples clasificaciones de usuarios en lugar de tener una clasificación general. Por ejemplo, es posible que ciertas zonas de la API sólo sean accesibles para algunos usuarios administrativos manejando el tercer nivel de seguridad.

Cuando se diseñan las políticas de seguridad de una API se deben considerar estos puntos:

1. **Con respecto a los programadores:**
   a) ***Existe algún módulo que se convierta en la API central o cada módulo tendrá su propia API?***
   b) ***El lenguaje utilizado tiene soporte para estándares como OAuth 2?***
   c) ***Si el usuario aún no existe en el sistema, cuáles son los pasos que debe seguir para darse de alta?***
   d) ***Los usuarios se pueden registrar a través de la API?***
   e) ***El registro a través de la API supondrá la necesidad de reducir la cantidad de información que se toma del usuario?***
   f) ***Cuál es la mínima información necesaria para poder administrar correctamente el comportamiento del usuario?***
2. **Con respecto a los administradores:**
   a) ***Los administradores tendrán acceso a las cuentas de usuario?***

b) ***Existirá una interfaz diferente para administrar la cuenta para los desarrolladores?***

c) ***Es posible medir la actividad del usuario en la API?***

d) ***El administrador tiene permiso de eliminar cuentas de usuario?***

e) ***Los usuarios tendrán acceso a la información de su cuenta a través de herramientas analíticas?***

Finalmente las recomendaciones generales en seguridad que se deben tomar son:

1. Utilizar cuando sea posible el protocolo HTTPS sobre HTTP cuando se envía información importante o privada, sobre todo si los clientes no contienen ninguna forma de cifrado.

2. Cuando la API soporte los métodos POST y PUT de HTTP es necesario realizar una serie de medidas dentro del servidor para validar que la información lleve el formato deseado[18].

3. Para las API que permiten la modificación de la información, no es suficiente realizar identificación sólo a través de la IP. Es posible hacer ataques de duplicado de IP lo cual permitiría a cualquiera acceder a nuestro sistema.

4. Utilizar llaves en las API sólo para información no crítica y sólo para acciones de lectura. Cuando se tiene una API pública es importante utilizar llaves que reconozcan entre mayúsculas y minúsculas. Esto es sencillo de aplicar y permiten asegurar un poco más las bases de datos al reducir el número de peticiones que tendrán respuesta.

   Esto también permite establecer una cuota de uso y analizar los datos de utilización de la API.

5. Se recomienda que las API utilizadas dentro de este proyecto deben de utilizar Oauth 2 para información proveniente de plataformas móviles o sistemas que vayan a modificar la información. Las ventajas de utilizar este protocolo sobre sistemas de nombre de usuario y contraseña o sistemas de sesiones lo hacen la elección correcta cuando se trata de manejar seguridad dentro de la API. Generalmente, las aplicaciones basadas en Oauth 2 muestran un formulario basado en web en lugar de la aplicación misma, lo que implica que la contraseña no es vista por la aplicación y por ende no es guardada en el dispositivo.

6. Dar soporte a otros tipos de sistemas de autenticación a pesar de que no se vayan a utilizar directamente dentro de las aplicaciones puede ayudar a escalar y conectarse con otros sistemas. Sin embargo, esta es una decisión que debe de tomarse en el momento que se necesite ya que actualmente la mayoría de las aplicaciones están basadas en Oauth 2.

[18] La información que se puede incluir en un ataque POST son bombas en las cabeceras, ataques de denegación de servicio, grandes cantidades de información desbordando el buffer, entre otros.

Conclusiones

El presenta trabajo de tesis concluyó con éxito cumpliendo con el objetivo general y siguiendo los estándares y la metodología escogida, al igual que se logró finalizar los requerimientos funcionales que se plantearon al inicio del proyecto. El objetivo general de este trabajo de tesis se logró exitosamente al demostrar que con las tecnologías propuestas es posible construir un sistema de monitoreo para el sistema de transporte público en la zona metropolitana del Valle de México. Estas tecnologías también permitirán tener una base para crear una plataforma de servicios orientados al monitoreo y la gestión de sistemas de transporte.

La metodología utilizada en el desarrollo de éste proyecto permitieron dividir en tareas manejables el sistema en su conjunto. Los primeros pasos hacia un proyecto de gran escala que posibilite la construcción de un mejor sistema de transporte han sido dados en este proyecto. Conocer y administrar una metodología de desarrollo adecuado permitió facilitar el desarrollo de este trabajo de tesis.

Se logró entender una parte de la problemática actual en los sistemas de transporte público y en cómo estos problemas afectan a los pasajeros y usuarios de estos servicios. Esto permitió un entendimiento más detallado de las necesidades del proyecto que se vieron reflejados en los capítulo en los que se diseñó.

La arquitectura presentada está basada en servicios web, utilizando la filosofía SOA, que permitió que el acoplamiento de servicios y módulos propios, así como de terceros, de una manera más sencilla facilitando la creación del prototipo al poder utilizar librerías externas.El realizar un análisis de las herramientas de software y hardware detallado permitirá que trabajos futuros puedan utilizar éste como base para plataformas más completas, con altas prestaciones de seguridad y gran velocidad de respuesta.

El beneficio que se obtenga de este estudio dependerá del seguimiento y aceptación tenga el proyecto final, así como el que éste genere en los sistemas de transporte y vialidades de la zona metropolitana. El trabajo en conjunto de los administradores, transportistas, directivos y organismos del gobierno relacionados con el transporte público es necesario para el mejoramiento de los sistemas de transporte a través de un mejor control del sistema en su conjunto. La investigación y la plataforma presentados en este trabajo de tesis pueden ayudar a cumplir ese objetivo.

A lo largo de la investigación también se estudió los requerimientos especiales para servicios internacionales (servicios ubicados en otros países que podrían trabajar en conjunto con

nuestro subsistema) y se encontró que se deben de considerar situaciones y características especiales que no se tomaron en cuenta en este trabajo de tesis como pueden ser retrasos debido al tiempo que le toma a los paquetes viajar a través de internet ,el hecho de que cada sistema puede manejar distintos protocolos o el problema que genera el trabajar con zonas horarias. En casos como estos existirán requerimientos especiales para cada servicio que se conecte posiblemente debido a factores políticos, sociales o culturales. Sin embargo esta distinción no se tomó en cuenta dentro de este trabajo ya que el proyecto está diseñado para tener una cobertura solamente regional.

La era de la información, el internet global y la reducción de los costos en el mundo de la electrónica han permitido que cada día existan más personas conectadas a los servicios web, lo que alienta a que cada vez más servicios sean trasladados a la nube. Los sistemas de transporte público no son la excepción y con la creciente necesidad que existe actualmente de dar un mejor servicio a cada vez más personas, la creación d sistemas de información en tiempo real y proyectos derivados serán cada vez más necesarios.

Finalmente, a partir de la experiencia obtenida a los largo del desarrollo de éste trabajo de tesis se pueden dar las siguientes recomendaciones para trabajos futuros:

1. Si se pretendiera utilizar este sistema para una infraestructura comercial, deben de utilizarse mapas profesionales como la versión empresarial de Google Maps o una versión personalizada de Open Street Map. La primera, al ser una versión pagada, incluye mejoras en los mapas digitales como señales de conducción, mejor señalamiento en el nombre de las calles y carreteras, indicaciones sobre sistemas turísticos, entre otros. Esta información se podría dar como un valor agregado para un enfoque de negocio.

2. Crear aplicaciones móviles que aprovechen esta información es un siguiente paso natural. Los dispositivos móviles se encuentran actualmente en todas partes y tienenun gran alcance de penetración dentro de la población.

3. Actualizar los paquetes, librerías, frameworks e intérpretes utilizados en el prototipo. Esto es principalmente cierto para el lenguaje de programación python que evoluciona a una gran velocidad.

4. Continuar con la utilización de la metodología Scrum y Kanban para el desarrollo del sistema a futuro. Al mismo tiempo este sistema puede ser reajustado para no tener una sobrecarga de organización y burocracia que termine dañando el proyecto a largo plazo.

5. Actualizar los dispositivos GPS a sistemas celulares que permitan el envío de más información además de la posición. Esto permitirá utilizar el estándar GTFS de una manera más completa.

6. El sistema puede adaptarse a otro tipo de sistemas AVL y utilizar otros medio de comunicación o inclusive utilizar otros protocolos de comunicación siempre y cuando se respeten

estándares internacionales.

7. La cantidad de datos dentro de la base de datos, así como la cantidad de información que debe ser procesada no crece linealmente conforme el número de transportes conectados al sistema aumenta. Por ello sería recomendable crear o rentar un sistema de almacenamiento en nube que escale horizontalmente conforme se necesite y utilizar un sistema NGIX que realice balanceo de carga en los servidores.

Apéndice A

Estudio comparativo entre diferentes servidores

El servidor utilizado para el desarrollo de la API dentro de éste proyecto es provisto por defecto con Django. Sin embargo no es un servidor que se recomiende para realizar aplicaciones de producción.

Realizar estudio de comportamiento de servidores es algo complicado, es necesario tomar en cuenta muchos factores que generalmente son difíciles de medir como la forma en la que se encuentran las estructuras de datos dentro de la memoria, que tan bien el servidor utiliza los tres niveles de memoria cache que el procesador trae, el lenguaje de programación principal del servidor, el protocolo de comunicación principal que utiliza el servidor para comunicarse con las aplicaciones y el qué tan bien se comporta el servidor en pruebas de estrés.

Es debido a todas las dificultades que se pueden encontrar dentro de la medición del comportamiento de los servidores que se decidió poner esto dentro de un apéndice. Es por ello que los resultados que se encuentran en esta sección deben ser tomados como una referencia y no como una medida final ya que no reflejan fielmente cómo se comportarían estos servidores dentro un ambiente de producción.

A.1. WGSI

WGSI es el acrónimo en inglés para Web Server Gateway Interface. Es una especificación para servidores web y servidores de aplicaciones, para comunicarse con aplicaciones web y servicios web. Es un estándar de Python descrito en PEP 333[49].

El objetivo es crea un interfaz que sea relativamente simple y comprensible que sea capaz de soportar la mayoría de las interacciones que existen entre un servidor web e internet.

Las aplicaciones WGSI pueden ser puestas en pila. Aquellas en la mitad de la pila se les conoce como aplicaciones intermediarias (middleware) y deben implementar ambos lados del la interfaz WGSI, tanto la de aplicaciones como la de servidores. Un servidor WSGI sólo recibe la petición por parte del usuario y se la pasa a la aplicación, después toma la respuesta de la aplicación y se la envía de nuevo al usuario. Estas son sus únicas funciones, todo aquello que se realiza con la información debe ser realizado por la aplicación o los aplicaciones intermediarias. No es obligatorio entender éste protocolo para poder utilizar servidores y aplicaciones que lo utilicen. Para realizar aplicaciones intermediarias es necesario tener un pequeño entendimiento de cómo hacer una aplicación con ambas interfaces a menos que utilicemos una aplicación que tenga soporte nativo para este protocolo.

Python soporta desde la versión 2.5 este protocolo de manera nativa. Para versiones anteriores puede ser instalado y para servidores en producción se puede utilizar Apache como mod_wgsi activado.

A.1.1. Servidor Web WGSI

El servidor debe de proveer dos cosas: un diccionario con variables de entorno y una función de *start_response*. El diccionario debe contener todos los valores de configuración para que el servidor funcione. La función *start_response* toma dos argumentos: *status* que contiene el código de estado HTTP como los explicados en el capítulo 5 y *response_headers* que es una lista de cabeceras HTTP para responder.

Es responsabilidad de la aplicación pasar estos dos argumentos al servidor.

A.1.2. Aplicación Final

Una aplicación es vista desde el servidor como una serie de métodos, clases u objetos que pueden ser ejecutados. Los argumentos a los métodos ___init___, ___call___, o la función inicial son: *environ* una serie de valores de entorno y *start_response* siendo un método ejecutable. *Aplicaciones intermediarias* Los componentes para una aplicación intermediaria deben de implementar ambas partes, tanto la interfaz del servidor web como la interfaz de la aplicación, y la eliminación de algunas restricciones. Las aplicaciones deben tan transparentes como sea posibles.

A.2. Ambiente de Prueba

El ambiente en el que se realizarán las pruebas de los servidores está constituido por los siguientes componentes

1. **Servidor:**
   a) 3GB de memoria RAM
   b) 2 core CPUs
   c) Zona Horaria America/Mexico_City
   d) Ubuntu 12.04 LTS (2013/07)
      1) Procesador AMD x64
      2) Linux 3.5.0-41-generic
   e) Modificaciones
      1) Aumento el límite a 10000 en /etc/security/limits.conf
      2) Actualización del archivo /etc/sysctl.conf para manejar un número mayor de conexiones TCP
      3) Modificación del archivo /etc/sysctl.conf de la variable

$$net.ipv4.tcp\_tw\_reuse = 1$$

2. **Redes:**
   a) Conexión a localhost a través de puerto 8080
   b) Interfaz de red interna 127.0.0.1
3. Interprete de Python 3.2
4. Motor de bases de datos PostgresSQL 9.1.9
5. Librería PostGIS 2.0.1 r9979
6. Librería GEOS 3.3.8-CAPI-1.7.8
7. Librería PROJ Rel. 4.7.1
8. Librería LIBXML 2.7.8
9. Sistema celular android 2.3 con GPS y conexión de datos Wi-Fi
10. **Servidores:**
    a) *Gevent 1.0rc1:* Librería de redes basada en corutinas que utiliza greenlet para proveer una API asíncrona sobre la librería libevent
    b) *uWSGI 1.9:* Este proyecto está diseñado para crear un servidor que permita construir aplicaciones de cualquier tipo sobre el. Servidores de aplicaciones, proxies, manejadores de procesos y monitores son implementados dentro de una misma API y un sistema de configuración común.

c) *Tornado 3.1.1:* Es un framework web escrito en Python sobre una librería asíncrona originalmente desarrollado por FriendFeed. Utilizando entradas no bloqueantes, el número de conexiones que puede manejar al mismo tiempo puede escalar a varios miles.

A.3. Análisis de comportamiento para diferentes servidores

Para realizar un análisis en el comportamiento que presentan algunos de los servidores se diseño un sistema de testeo muy simple. Idealmente, un estudio de servidores se realizaría bajo ambientes diseñados para ellos como aplicaciones no triviales como la que se utiliza aquí. Sin embargo, debido a la complejidad de realizar un estudio de esta manera y que esto se encuentra fuera del objetivo de la tesis se dejará este análisis como una ayuda para futuras referencias.

Para este análisis se utiliza AutoBench para generar tráfico realista y Motor para crear implementaciones de entradas no bloqueantes y trabajar con Tornado. También es utilizado Pymongo para realizar entradas no bloqueantes con Gevent.

Para cada prueba se realizaron peticiones directamente contra el servidor. Las variables de entorno son min_rate=10 y max_rate=2000 que significan el número de peticiones que se harán al servidor por segundo. Todas las peticiones fueron sobre una sola instancia, por lo que no se toma en cuenta el tiempo que se pierde al hacer balanceo de carga o distribución de trabajos en clusters.

El servidor Gevent se utilizará como punto de comparación para los otros dos servidores. Los datos de cada una de las mediciones son guardados en un archivo de texto y dibujados en gráficas utilizando la librería de Javascript Flot.

El primer análisis que se realiza es la comparación entre dos servidores muy utilizados dentro del mundo Python: Tornado y Gevent.

A.4. Tornado y Gevent

Los servidores tienen el manejo de sesiones y la opción de debug desactivadas. Para realizar una comparación lo más justa posible, no se realizaró ningún tipo de optimización avanzada en los servidores.Para todas las pruebas sólo se realiza una petición por conexión sin manejo de keep-alive por parte de los servidores.

Para esta prueba se compara una implementación de la API realizada en este proyecto utilizando implementaciones wgsi de cada uno de los servidores. En la figura A.1 se muestra una toma de pantalla de las gráficas que obtuvieron como resultado. En las figuras A.2, A.3 y A.4

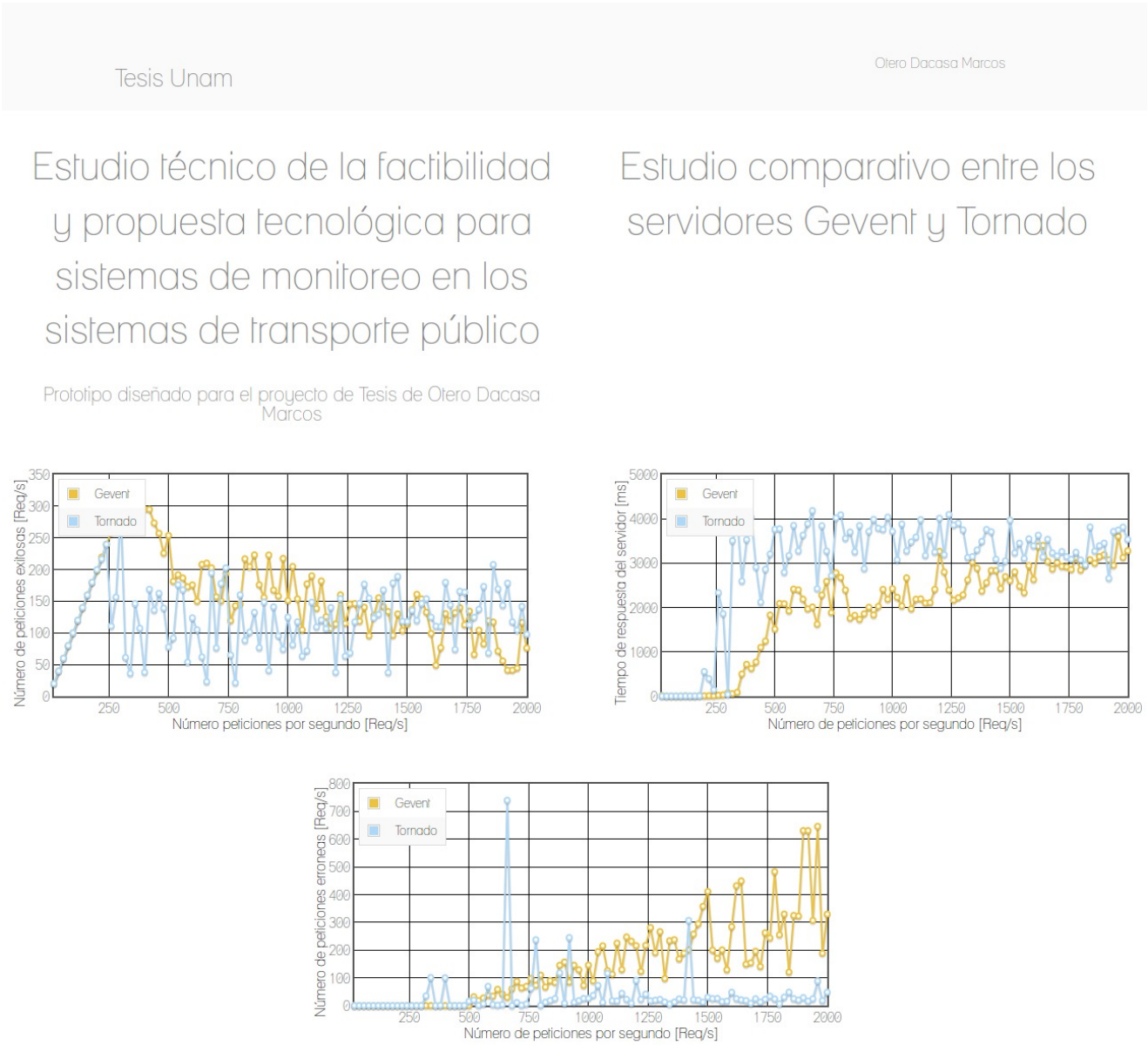

Figura A.1: Gráficas del comportamiento de los servidores Gevent y Tornado

se puede ver un acercamiento a cada una de las gráficas obtenidas.

A partir de las respuestas se podría deducir que en condiciones no óptimas Gevent resulta ser más veloz que Tornado. Esto puede ser gracias a la librería greenlet, libev y a monky-patching que le dan un sistema de programación asíncrono liberando el servidor de sobrecargas.

Otra razón por la que este framework tuvo un comportamiento menor puede ser que Tornado fue diseñado para manejar un alto número de conexiones persistentes y aquí se utilizó negociación de conexión por cada petición GET que se se hacía. Sin embargo, se debe notar que

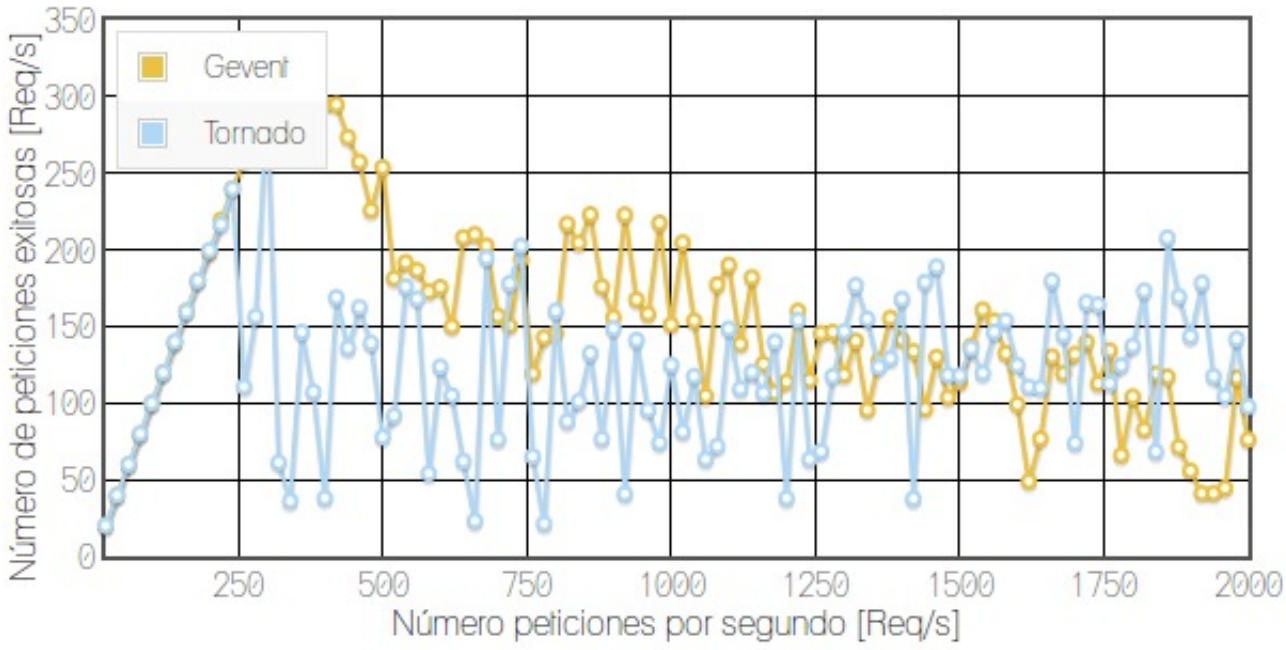

Figura A.2: Gráfico del número de peticiones exitosas Tornado y Gevent

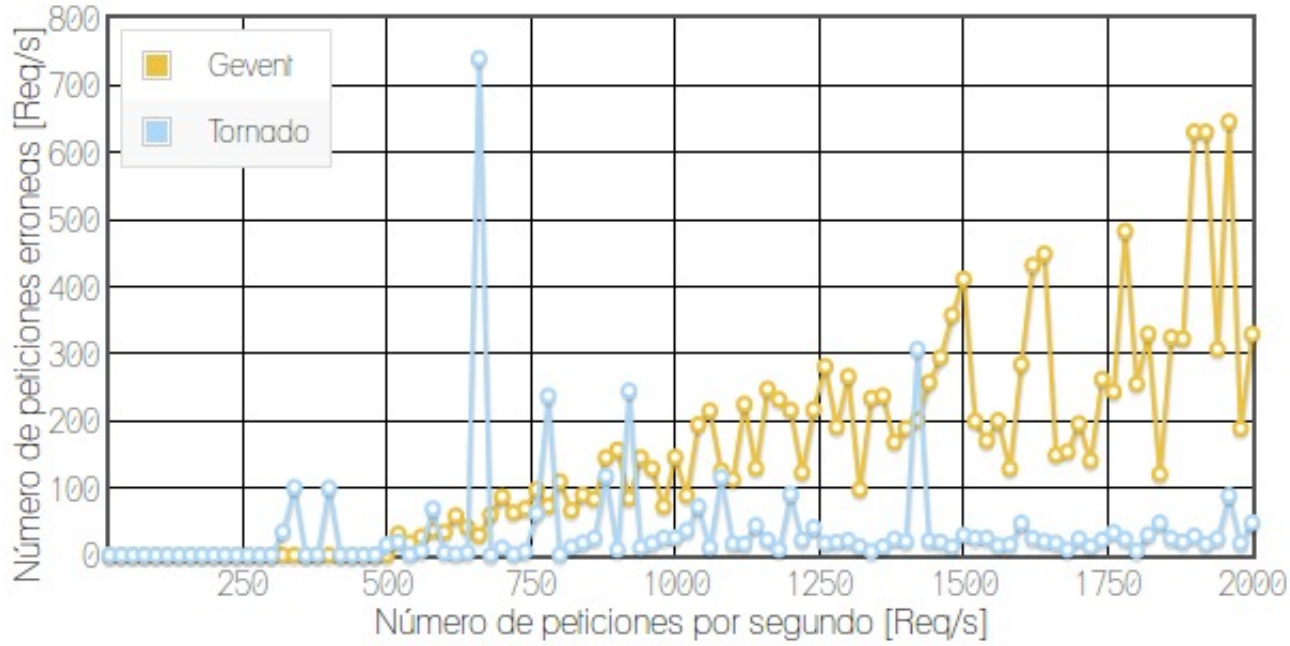

Figura A.3: Gráficas del número de peticiones erróneas Tornado y Gevent

parece ser que Tornado puede manejar un mayor número de conexiones al mismo tiempo de una manera más estable como se nota en la figura A.3 y también aumenta su velocidad de respuesta.

Por ahora el servidor Gevent parece ser la mejor opción para diseñar aplicaciones web que requieren manejar una cantidad variable de usuarios conectados en un instante y que requieren una buena velocidad de respuesta. Para el segundo análisis se tomará una comparación entre

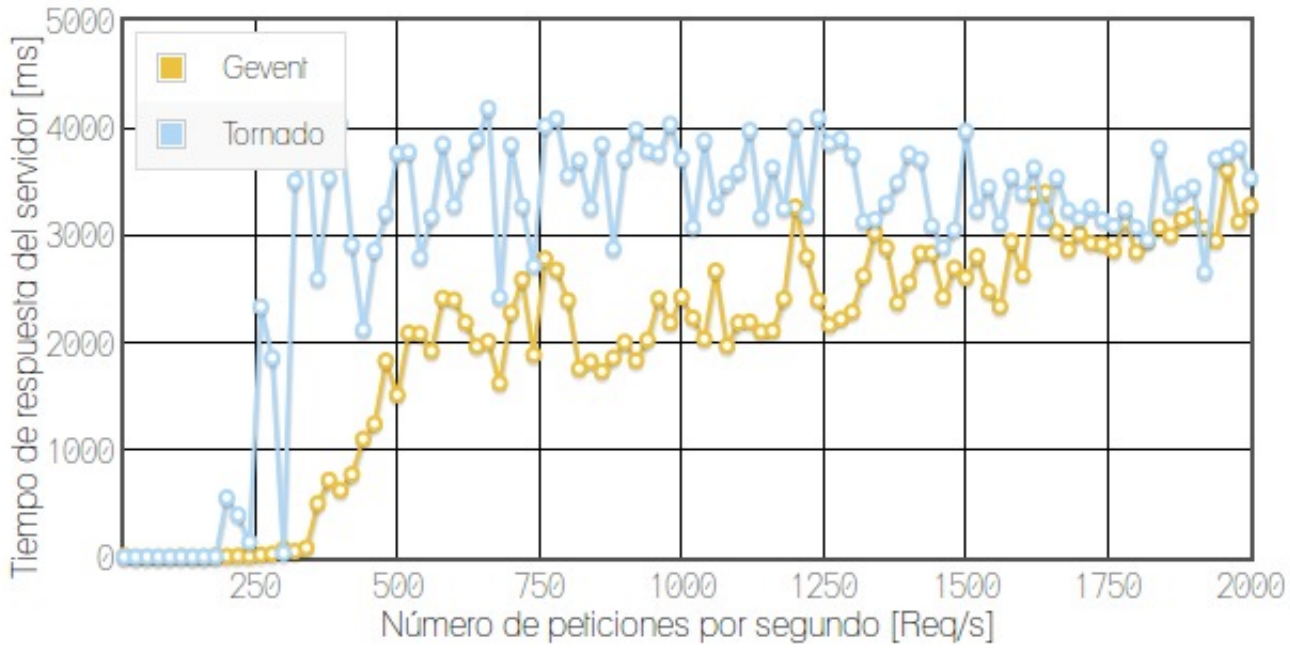

Figura A.4: Gráficas del tiempo de respuesta de los servidores

los servidores Gevent y el servidor uWSGI+Gevent.

A.5. uWSGI+Gevent y Gevent

Las configuraciones utilizadas en éste análsis son las mismas. La linea de comando que ejecuta el servidor uWSGI es

uwsgi –http-socket :8080 –file gtfs_test.py –gevent 2000 -l 1000 -p 1 -L

En la figura A.5 se muestra una toma de pantalla de las gráficas que se obtuvieron como resultado

Figura A.5: Gráficas del comportamiento de los servidores Gevent y uWSGI

En las figuras A.6, A.7 y A.8 se puede ver un acercamiento a cada una de las gráficas obtenidas.

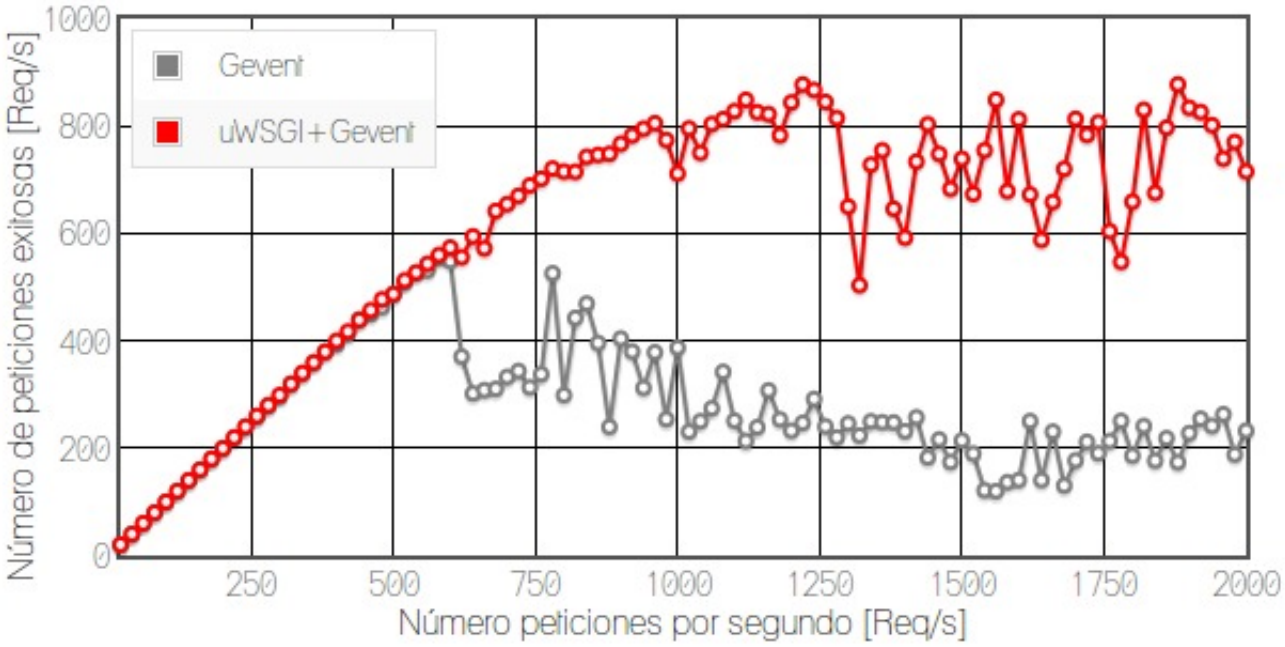

Figura A.6: Gráfico del número de peticiones exitosas uWSGI y Gevent

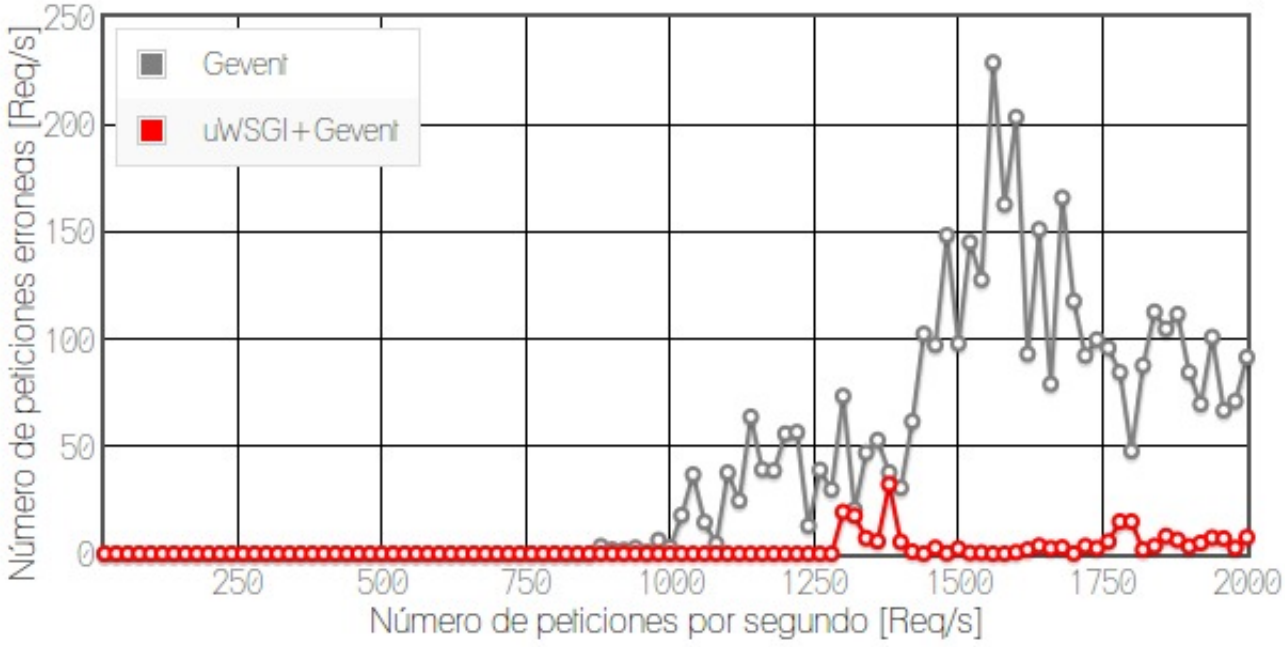

Figura A.7: Gráficas del número de peticiones erróneas uWSGI y Gevent

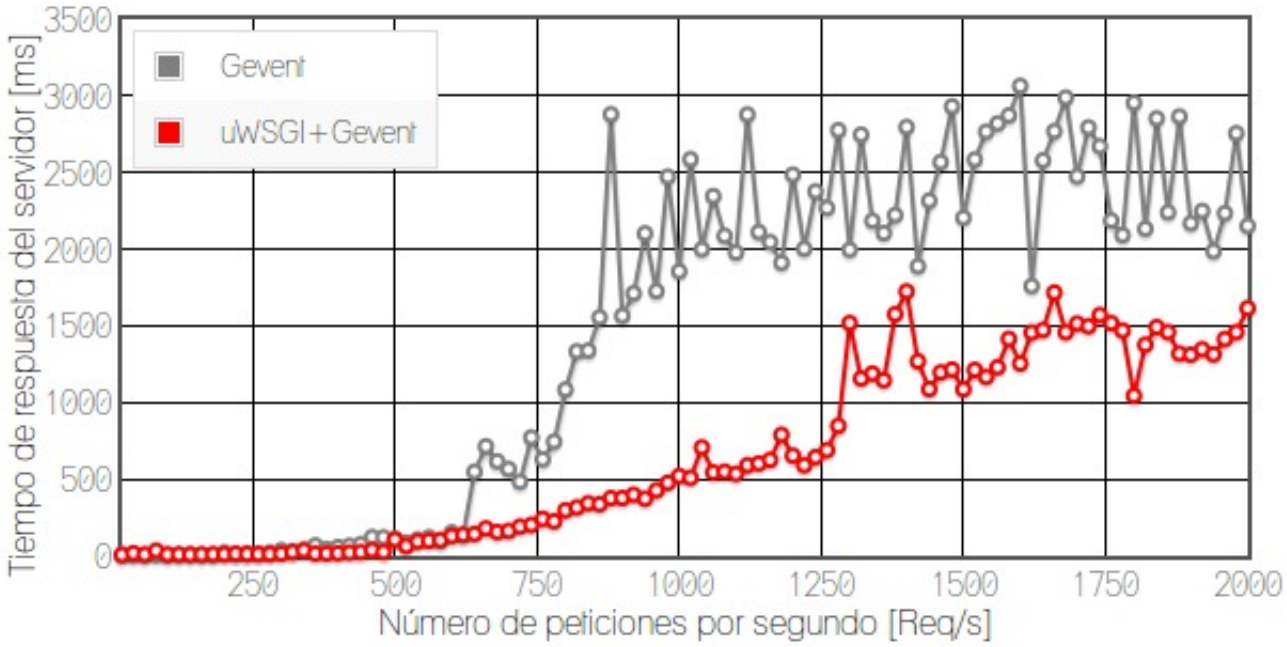

Figura A.8: Gráficas del tiempo de respuesta de los servidores

Como se puede ver existe una mejora notable cuando se agrega a nuestro servidor el sistema de aplicaciones uWSGI que tiene una respuesta mucho mejor que la que tenía el servidor Gevent por sí mismo.

Aunque este tipo de estudios no son definitivos, se espera que sean de ayuda en la elección de la tecnología de servidores. A pesar de ello, estudios como éstos deben de llevarse a cabo dentro de los servidores finales y utilizando las librerías y sistemas operativos finales.

Existen una gran cantidad de servidores que se podría analizar pero que no se cubrirán en éste trabajo:

1. **Gunicorn:** El servidor Gunicorn nace de un servidor llamado Unicorn escrito en el lenguaje Ruby. El objetivo de éste proyecto es diseñar un sistema ligero mientras proveen balance entre velocidad de respuesta y facilidad de uso.

2. **Chaussettes:** Es un servidor WSGI que se puede utilizar para correr aplicaciones web. Su particularidad es que puede utilizar un socket como cualquier otro servicio o utilizar sockets abiertos.

3. **Circus:** Puede crear sockets y manejarlos como procesos. La idea principal detrás de éste proyecto es que un proceso hijo del proceso central puede heredar todos los descriptores de archivos abiertos.

Para Chausettes y Circus existe un estudio de comportamiento en donde se realizan un análisis muy similar al que se presentó en este apéndice.[53][54]

Una recomendación basado en la información obtenida aquí sería utilizar uWSGI + Gevent como servidor final. Aunque es necesario hacer estudio personalizados en la plataforma final sobre la que se va a trabajar. ApéndiceB

Apéndice B

Estimación de costos

La estimación de costos de un sistema informático es un tema complicado de estudiar. Existen una gran cantidad de factores que se deben tomar en cuenta y la mayoría no se pueden saber hasta que no se tiene el proyecto final como, por ejemplo, el salario de un programador que trabaje en el proyecto. En realidad ni siquiera las grandes empresas de la informática pueden saber cuánto les costará a ciencia cierta.

La primer dificultad se encuentra en el servidor, como se planteó en el Apéndice A existen una gran cantidad de servidores distintos y con funcionalidades distintas (recomendaciones, balanceo de carga, API, página web) que se pueden encontrar en la nube o ser privados, diferentes tipos de representación de datos como se analizó en el capítulo 5 y cada una genera una cantidad de bytes de salida distinto.

La segunda dificultad es el costo de la energía eléctrica. Los sistemas eléctricos tienen costos que dependen de la empresa en la que se esté contratando, la época del año, la fuente de la energía (la energía solar es más costosa que la energía proveniente del petroleo en éste momento) y el consumo anual. El consumo depende de la infraestructura de telecomunicaciones que se tenga, el consumo de los procesadores y memorias (Entre más memoria consume un proceso, también consume más energía), la cantidad de ciclos de reloj de procesador que está accediendo, e inclusive cuestiones difíciles de medir como si la información está siendo guardada en la cache del procesador o si éste debe ir a buscarla a la memoria.

La tercera dificultad es el costo por Gigabit de ancho de banda consumido y esto está relacionado con el número de usuarios en un instante dado (otro factor que no se puede prever) y la cantidad de información que cada usuario obtiene. Para un sistema que utiliza toda su capacidad el costo por Gigabit por hora será mejor para uno que utiliza la mitad de su capacidad.

Existen más dificultades que hacen que una medida exacta sea muy compleja de obtener, es por ello que en ésta sección se da un estimado del costo utilizando medidas que se tienen al

momento de realizar éste trabajo.

1. **Costos de almacenamiento:** Existen una gran cantidad de operadores en la nube que ofrecen grandes capacidades de almacenamiento por un costo muy pequeño. En promedio el costo es de US$0.025 por GB.

2. **Costos de distribución de información:** El envío de la información desde la base de datos hasta los usuarios ocupa una pequeña porción de ancho de banda de salida. Empresas como Amazon Web Services[1] cobran por cada GB de salida US$0.12 después del primer GB.

3. **Costos de servidores:** Este costo incluye los costos por hosting y algunas funcionalidades que se dan como una IP pública, etc. Este número es muy complicado de obtener pero si se toman los precios de Amazon Web Services, utilizando la estancia Micro Reserved Instance, cuesta US$0.02 por hora.

Cada una de las peticiones al servidor retorna en promedio 200Kb de información en formato JSON. Suponiendo el peor de los casos en el que un usuario que hace un petición cada hora y existe el cobro de GB por salida y una base de datos que con un mensaje del formato GTFS que pesa 500Kb (Junto con la información acerca de la agencia entre otros), el sistema costaría por hora:

$$C_{US} = 0{,}02 + 0{,}12\left(\frac{200[Kb]}{1x10^6[Kb]}\right) + 0{,}025\left(\frac{500[Kb]}{1x10^6[Kb]}\right)$$

$$C_{US} = \$0{,}0200365[Usuario]$$

Suponiendo un valor de cambio US$1 = MX$13

$$C_{MX} = \$0{,}2604$$

Para el mejor de los casos se tendría:

$$C_{US} = 0{,}02 + 0{,}025\left(\frac{500[Kb]}{1x10^6[Kb]}\right)$$

$$C_{US} = \$0{,}0200125[Usuario]$$

$$C_{MX} = \$0{,}2601625$$

Si se atienden un millón de usuarios y sólo se contemplan estos gastos, el costo por hora estaría entre

$$\$260,162{,}5 < C_{MX} < \$260,400$$

Estos resultados no deben tomarse como mediciones finales debido a todos los factores que no se toman en cuenta como teoría de colas, el que los usuarios no usarían el sistema entre ciertas horas (es una de las ventajas de que el sistema sólo funcione en un país) o que se pueden reducir precios a través de contratos, comprimiendo la información de salida, etc.

[1] http://aws.amazon.com/ec2/?navclick=true

Bibliografía

[1] *Ley de Transporte y Vialidad del Distrito Federal*. Publicada en la Gaceta Oficial del Distrito Federal el 26 de Diciembre de 2002

[2] *Plan Integral de Transporte y Vialidad 2007 - 2012*. Publicada en la Gaceta Oficial del Distrito Federal el 22 de Marzo de 2010

[3] Instituto de Políticas para el Transporte y el Desarrollo, *Perspectivas de crecimiento de la Red Metrobús y transporte integrado del Distrito Federal a 2018*

[4] Nicolai M. Josue (2007) *SOA in Practice, The Art of Distributed System Design*, 1a Edición, Estados Unidos.

[5] Qi Yu, Athman Bouguettaya (2009) *Foundations for Efficient Web Service Selecton, Foundations for Efficient Web Services*, 1a Edición, Ed. Springer, Estados Unidos.

[6] Douglas K. Barry (2013) *Web Services, Service-Oriented Architectures, and Cloud Computing*, 2a Edición, Ed. Morgan Kaufman, Estados Unidos.

[7] L. G. Meredith (2012) *Design Patterns for the Web*, PrePrint Edition, Ed. Artima Pressm, Estados Unidos.

[8] G. Radha Mani & G.S.V Radha Krishna Rao (2007) *Web Services Security and e-business*, 1a Edición, Ed. Idea Group Publishing, Reino Unido.

[9] Richard Ferraro, Murat Aktihanoglu (2011), *Location aware application*, 1a Edición, Ed. Shelter Island, Estados Unidos.

[10] Maigua Gustavo, López Emanuel (2012), *Buenas prácticas para l̃a dirección de proyectos informáticos*, 1a Edición, Ed. Edutecne, Argentina.

[11] Gershenson C, Pineda LA (2009) *Why Does Public Transport Not Arrive on Time? The Pervasiveness of Equal Headway Instability*. PLoS ONE 4(10): e7292. doi:10.1371/journal.pone.0007292

[12] Gershenson C (2011) *Self-Organization Leads to Supraoptimal Performance in Public Transportation Systems*. PLoS ONE 6(6): e21469. doi:10.1371/journal.pone.0021469

[13] Vincent Blondel and Nicolas de Cordes, Adeline Decuyper, Pierre Deville, Jacques Raguenez, Zbigniew Smoreda (2013) *Mobile Phone Data for Development: Analysis of mobile phone datasets for the development of Ivory Coast* 1a Edición, Ed. NetMob.

[14] Jorge Fernández González (2011), *Introducción a las metodologías ágiles*, 1a Edición, Ed. UOC, España

[15] Fielding, Roy Thomas. (2000) *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University of California, Irvine, 2000. Revisado Junio, 2013, desde <http://www.ics.uci.edu/̃fielding/pubs/dissertation/top.htm.>

[16] WHO (2010), *Urban Health*, Revisado Junio (2013), desde <http://www.who.int/gho/urban_health/situation_trends/urban_population_growth_text/>

[17] FMI (2012), *Mexico, Total population 2013* Revisado Junio (2013), desde <http://www.statista.com/statistics/19308/total-population-in-mexico/>

[18] INEGI (2012), *México en cifras*, Revisado Junio (2013), desde <http://www3.inegi.org.mx/sistemas/mexicocifras/default.aspx?e=9>

[19] Google (2013), *Referencia de la Especificación general de feeds de transporte público*, Revisado Julio, 2013, desde <https://developers.google.com/transit/gtfs/reference>

[20] SETRAVI (2012), *Base de Datos Abiertos de Transporte*, Revisado Julio (2013), desde <http://www.setravi.df.gob.mx/wb/stv/datos_abiertos_de_transporte_de_la_ci u d a d _ d e_mexic>

[21] Google (2013), *Protocol Buffes*, Revisado Julio, 2013, desde <https://developers.google.com/protocol-buffers/docs/overview>

[22] Django Project (2013), *Documentación oficial del proyecto Django*, Revisado Junio, (2013), desde <http://www.djangoproject.com>

[23] Tornado Web Framework (2013), *Tornado 3.1.1 documentation*, Revisado Junio, (2013), desde <http://www.tornadoweb.org/en/stable/>

[24] Mysql (2003), *D.4. Changes in release 3.23.x (Recent; still supported)*, Revisado Junio, (2013), desde <http://dev.mysql.com/doc/refman/4.1/ja/news-3-23-x.html>

[25] Mysql (2013), *MySQL 5.6 Reference Manual* , Revisado Julio, (2013), desde <http://dev.mysql.com/doc/refman/5.6/en/index.html>

[26] Microsoft (2013), *SQL official documentation*, Revisado Julio, (2013), desde <http://www.microsoft.com/en-us/sqlserver/default.aspx>

[27] Postgres (2013), *PostgresSQL 9.3.0 Documentation*, Revisado Julio, (2013), desde <http://www.postgresql.org/docs/9.3/interactive/index.html>

[28] BistonGIS (2012), *Compare SQL Server 2008 R2, Oracle 11G R2, PostgresSQL/PostGIS 1.5 Spatial Features*, Revisado Julio, (2013), desde <http://www.bostongis.com/? content_name=sqlserver2008r2_oracle11gr2_postgis15_compare#221>

[29] PostGIS (2012), *PostGIS Cheatsheet*, Revisado Julio, (2013), desde <http://www.postgis.us/downloads/postgis20_cheatsheet.html>

[30] Yahoo! (2010), *Yahoo! Cloud Serving Benchmark*, Revisado Julio, (2013), desde <https://docs.google.com/gview?url=http://research.yahoo.com/files/ycsb-v4.pdf>

[31] Open Geospatial Consortium (2013), *OGC Standards*, Revisado Julio, (2013), desde <http://www.opengeospatial.org/standards>

[32] Open Source Geospatial Foundation (2013), *open source Geospatial Foundation Tools* , Revisad Julio, (2013), desde <http://www.osgeo.org/>

[33] ISO (2008), *ISO 4217 Currency Codes*, Revisado Julio, 2013, desde <http://www.xe.com/iso4217.php>

[34] Open GPS Tracker (2009), *Open GPS tracker documentation*, Revisado Julio, 2013, desde <http://opengpstracker.org/>

[35] W3C (2013), *Universal Resource Identifiers: Recomendations*. Revisado Agosto, 2013, desde <http://www.w3.org/Addressing/url/4_URI_Recommentations.html.>

[36] TimezoneDB (2013), *List of timezones*, Revisado Agosto 2013, desde <http://timezonedb.com/time-zones>

[37] ISO (2008), *Codes for the reprensentations of names of languages*. Revisado Agosto, 2013, desde <http://www.loc.gov/standards/iso639-2/php/code_list.php.>

[38] W3C (2009), *Lenguajes utilizados en HTML y XML*, Revisado Agosto (2013), desde <http://www.w3.org/International/articles/language-tags/>

[39] Apache Project (2008), *Versioning Numbering Concepts*. Revisado Agosto, 2013, desde <http://apr.apache.org/versioning.html>

[40] W3C (2000), *Techniques For Accessibility Evaluation And Repair Tools*. Revisado Agosto, 2013, desde <http://www.w3.org/TR/AERT#color-contrast>

[41] National Geospatial Intelligence Agency (1997), *DoD World Geodetic System 1984*, Revisado Agosto, 2013, desde <http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html>

[42] Bemers-Lee, Tim, Roy T. Fielding, et al. (2005), *Uniform Resource Identifier (URI): Generic Syntax*, RFC 3986, Revisado Agosto, 2013, desde <http://www.rfc-editor.org/rfc/rfc3986.txt>

[43] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Mastinter, P. Leach, and T. Berners-Lee. (1999), *Hypertext Transfer Protocol – HTTP/1.1*, Revisado Septiembre 2013, desde <http://www.w3.org/Protocols/rfc2616/rfc2616.txt>

[44] D. Crockford (2006), *The application/json Media Type for JavaScript Object Notation (JSON)*, Revisado Septiembre 2013, desde <http://www.ietf.org/rfc/rfc4627.txt>

[45] Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler, Francois Yergeau (2008), *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, Revisado Septiembre 2013, desde <http://www.w3.org/TR/REC-xml/>

[46] IANA (1996), *Multipurpose Internet Mail Extensions*, Revisado Septiembre 2013, desde <http://tools.ietf.org/html/rfc2046>

[47] IANA (1996), *Mime Media Types*, Revisado Septiembre 2013, desde <http://www.iana.org/assignments/media-types>

[48] WSGI (2013), *Web Server Gateway Interface*, Revisado Octubre 2013, desde <http://wsgi.readthedocs.org/en/latest/index.html>

[49] Python (2013), *Python Web Server Gateway Interface v1.0*, Revisado Octubre 2013, desde <http://www.python.org/dev/peps/pep-0333/>

[50] Python (2013), *The Hitchhiker's Guide to Python*, Revisado Septiembre 2013, desde <http://docs.python-guide.org/en/latest/>

[51] Internet Engineering Task Force (2012), *The OAuth 2.0 Authorization Framework*, Revisado Octubre, 2013, desde <http://tools.ietf.org/html/rfc6749>

[52] Jakob Nielsen, Don Norman (2013) The Definition of User Experience, Revisado Octubre 2013, desde <http://www.nngroup.com/articles/definitionuserexperience/>

[53] Tarek Ziadek (2010), *WSGI WSGI Web Servers Bench*, Revisado Octubre, (2013), desde <http://blog.ziade.org/2012/06/28/wgsi-web-servers-bench/>

[54] Tarek Ziadek (2010), *WSGI Web Servers Bench Part 2*, Revisado Octubre, (2013), desde <http://blog.ziade.org/2012/07/03/wsgi-web-servers-bench-part-2/>

Índice alfabético