MapViz: A Framework for Visualization of Floating Car Data

SIAMAK SADEGHIANFAR

ROYAL INSTITUTE
OF TECHNOLOGY

Master of Science Thesis
Stockholm, Sweden 2012

MapViz: A Framework for Visualization of Floating Car Data

SIAMAK SADEGHIANFAR

Master's Thesis in Engineering and Management of Information Systems
at Department of Transport Science
Royal Institute of Technology, year 2012
Supervisor was Harilaos Koutsopoulos
and Mahmood Rahmani

Royal Institute of Technology
Department of Computer and Systems Sciences
Forum 100, SE-164 40 Kista, Sweden

MapViz: A Framework for Visualization of Floating Car Data

Abstract

Visualization is representation of data in a visual form that is more comprehendible. The main purpose of visualization is to communicate information effectively while balancing design with functionality. There are a variety of applications for visualization in the field of traffic such as CADD applications, traffic simulation and transit planning. One of the scenarios that visualization can be of great help is Floating Car Data. FCD refers to the data collected through a highly population sensor network. These sensors can be in form of stationary sensors installed on the road or GPS-equipped vehicles traveling the road network and broadcasting their coordinates. This data is of tremendous value for travel planning, monitoring and management.

MapViz is a visualization platform designed as a part of this thesis that provides the infrastructure for consuming digested FCD and visualizing the result on a map based on specific scenarios. The scenarios under focus in this work are real-time FCD visualization and city travel planning for suggesting routes between two locations.

MapViz is divided into two applications with independent responsibilities. MapViz Data Service consumes the FCD resulted after a map matching process. The data gets aggregated, refined, enriched and then transformed to a more suitable data format in Data Service and can be retrieved through a REST API in a well-defined JSON format. MapViz Visualizer is responsible for consuming the transformed data and rendering it as an overlay on top a map as well as handling interactions with the user.

MapViz is built on top of third-party services such as a map provider and geocoding services. A series of abstracts are used to ensure loose coupling and vendor neutrality in the design of MapViz, which enables the platform to switch between third-party services with trivial effort.

Acknowledgement

I would like to thank my supervisors, Harilaos Koutsopoulos and Mahmood Rahmani, at department of ABE at KTH who supported this work with their invaluable feedback. Furthermore, I thank Traffic Stockholm for providing the FCD used in the context of this work as well as NAVTEQ for providing the digital road network of Stockholm.

Table of Contents

Figures

Tables

1. Introduction

This thesis is based on a work done at Department of Transport Science, Division of Traffic and Logistics, Royal Institute of Technology to deliver a map-matching and path inference component that maps low frequency Floating Car Data (FCD) received in the Stockholm area to the digital road network of Stockholm city(Koutsopoulos, et al., 2010).

This thesis is a starting point for a designing a web-based platform called MapViz which is capable of digesting and visualizing geographic data without binding itself to any particular provider or vendor. This chapter describes the background of this work and focuses on explanation of concepts like visualization, FCD and how they are related in this thesis. Later in this chapter the objectives are described.

1.1. Visualization

Data visualization is science of visual representation of data, which means "information which has been abstracted in some schematic form, including attributes or variables for the units of information" (Friendly, 2009). According to Friedman, the main goal of data visualization is “communicating information clearly and effectively”(Friedman, 2008). This doesn’t necessarily entail lack of elegance in favor of functionality or exaggerated sophistication for beauty purposes. Aesthetic and functionality are two aspects of data visualization that go hand in hand to communicate key aspects of an often sparse complex set of data in a more intuitive way. The balance of design and function is an important part of data visualization that is often left out and leads to a failure to serve the purpose.

Research and publication in visualization defines two subfields: Scientific Visualization and Information Visualization (Friendly, 2009). A third subfield, visual analytics is emerging. Scientific visualization produces visual displays of 3D phenomena associated with scientific processes such as bonding of molecules in computational chemistry. Information visualization generally applies to the visual representation of large-scale collections of non-numerical data such as network of relations on the Internet. Visual analytics focuses on providing visual interfaces to explore analytical data in response to terrorist attacks and natural disasters (Rhyne, 2007). All three subfields of visualization are applicable to transportation and can provide valuable insight.

Although visualization is gaining popularity in the recent years, it is fairly misunderstood due to lack of standards and guidelines available to the transportation professionals on the use and benefits of visualization (Hixon, 2006). Nevertheless, Hixson sees a movement to increase the use of visualization in the field of transportation. Applications of visualization in transportation includes:

Computer-Aided Drafting and Design (CADD) Applications

Software products such as AutoCAD are used to create three-dimensional renderings of buildings and roads, which increases the level of understanding of the finished state of a potential project.

Animation and Simulation Software Applications

With the advances in animation techniques, the transportation community has been able to take advantage of simulation in several important projects. An early example of computer-assisted visualization was the Border Wizard project, which aimed at illustrating the trade-off between productivity and homeland security issues (Caldwell, 2002). The intended audience was a diverse group of people involved in logistics and national security. Visualization was perhaps the only technique that could bring such a diverse group to agree on how the system should operate.

Another example is the output screens of TRANSIMS, a micro-simulation modeling strategy that visualizes second-by-second movement of travelers in a three dimensional environment (Lawson & Jonnalagadda, 2006).

Other advances in simulation applications include driving simulators, terrestrial, mobile and airborne LIDAR[1] surveying systems and temporal visualization using RFID[2] (Manore, 2008).

Data Visualization in Transit Planning

Traditionally internal problems with transit data have prevented effective use of visualization in this area. By introduction of general purpose RDBMS[3], GIS[4] software and standard data models, majority of these issues have been resolved and it has become possible to visualized performance transit data (Kimpel, 2007). Tri-Met, the regional transit agency for Portland, Oregon Metropolitan Area, has developed applications that makes it possible to query and visualized a large amount of data

[1] Light Detection And Ranging
[2] Radio Frequency Identification
[3] Relational Database Management System
[4] Geographic Information System

collected through Automated Passenger Counter (APC) and Automated Vehicle Location (AVL) technologies. As an example, Figure 1 shows weekly on-time performance at time points for a particular route as a series of pie charts.

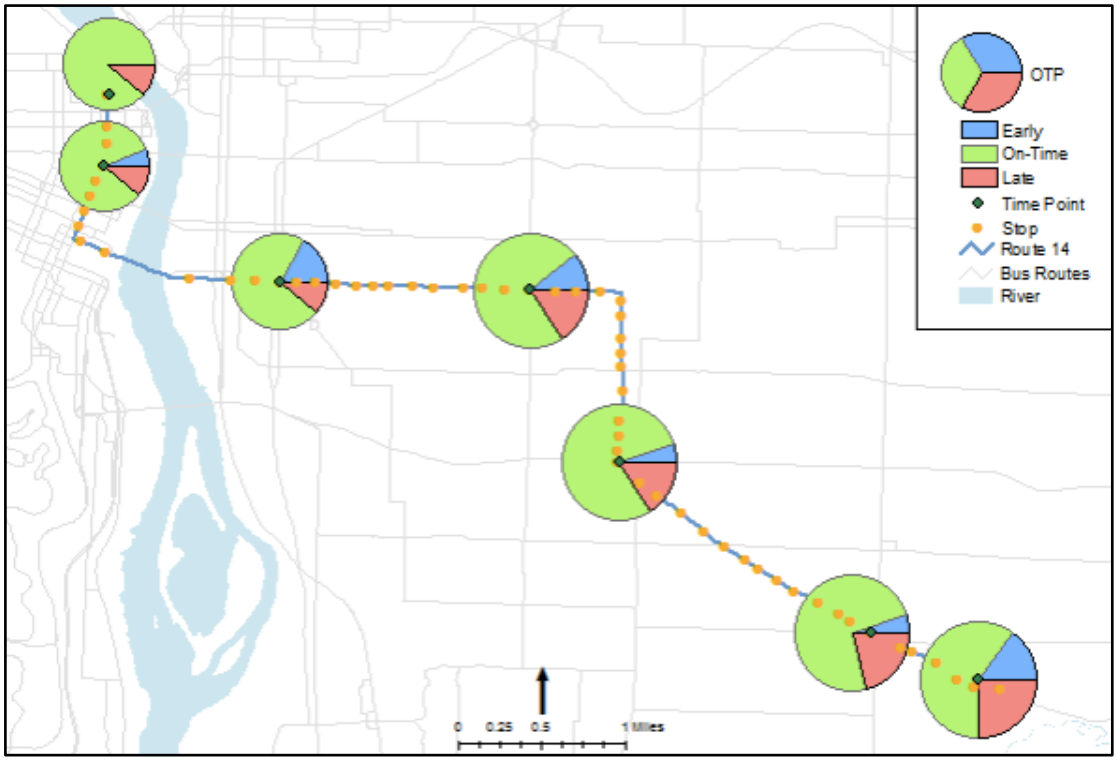


Figure 1 – On-time performance visualization by stop location, courtesy of (Kimpel, 2007)

Kimpel (2007) furthermore suggests moving away from one-off projects and incorporate long-term performance monitoring for transit operations by generating on-the-fly graphic outputs for the complete system. These applications can

Data Visualization for Understanding Freight

The volume of freights has been consistently growing over the years. The transportation systems must accommodate not only the increase in the volume but also an increased need for more timely and reliable shipments. Visualization of freight movements can be used to gain more insight into the complex freight systems and design effective solutions to meet the demand in the growing world of transportation (Schmitt et al., 2007).

4-Dimensional Visualization for Dynamic Interactions

Taking advantage of the advancements in the computer gaming industry, it is possible to move from static 3D rendering to 4D visualization systems that are more powerful in depicting dynamic interactions. The 4D visualization incorporates time as the fourth dimension to provide more accurate situational awareness and let the user "fly through a virtual scene as it unfolds in real-time" (Howard et al., 2008).

Another example is the animations of road conditions developed in the CATT Lab to visualize a variety of data stream including point sensor data, incident data, incident response land closures, dynamic message signs, weather and more (Pack et al.,

2007). In a similar effort, Oak Ridge Laboratories have produced aviation visualizations that depict flight proximities bases on standard government guidelines and proposed flight plan adjustments (Rose, 2005).

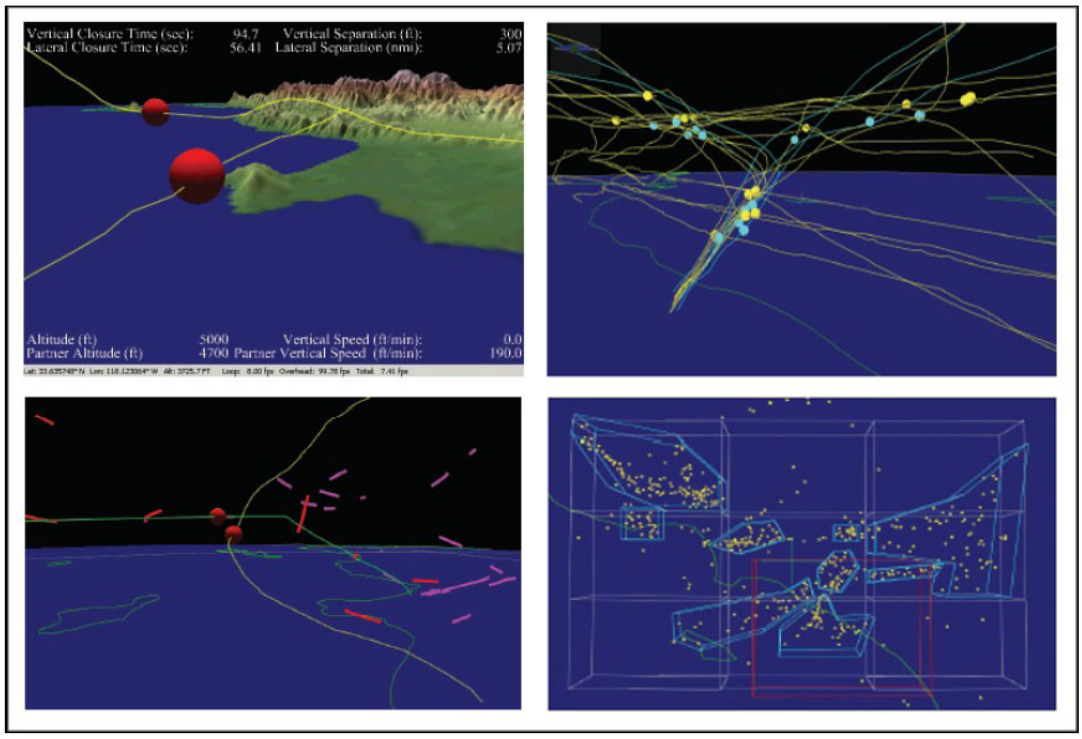


Figure 2 - Aviation Visualization, courtesy of (Rose, 2005)

1.2. Floating Car Data

Floating Car Data (FCD) refers to the data collected from a highly distributed sensor population for example sensor-equipped vehicles traversing the road network. This data may include among other things the location and speed of the vehicle, road hazards, emission, and weather conditions. The collected data is then aggregated at a processing center and valuable traffic information is deduced from it, which can be redistributed back to travelers.

One of advantages of using FCD in comparison to the traditional methods is network coverage. Mobile sensors can move everywhere in a road network and report traffic related data, as in contrast stationary sensors should be installed in a specific location. The other advantage of FCD is being cheaper compare to stationary sensors where you need to install a gantry, or dig in the ground. If a mobile sensor fails you can still get data from other sensors, but in case of stationary sensors, that's not possible.

Despite the advantages of FCD, it might also arise some privacy issues due to permanent traceability of the drivers (Rass et al., 2008). These concerns can somewhat be mitigated by anonymimzation of the drivers identity.

1.2.1. Data Collection

There are a number of different methods for collecting, processing and redistributing floating car data. Kerner et. al (2005) created a traffic information sharing system incorporating a client/server architecture. The server propagates speed values with threshold to the vehicles in the network. Vehicles in return send and update back to server, whenever their measured speed on the road differs from the advertised value by an amount larger than the threshold.

An alternative approach has been suggested using a mobile P2P architecture under the name Grassroots (Goel et al., 2003). In this prototype, each vehicle is equipped with a GPS sensor and broadcasts its velocity to other vehicles in the network using a wide range peer-to-peer communication method.

Several technical solutions are suggested in regard to vehicle behavior, algorithms and also aspects of communicating to the traffic centers. Figure 3 demonstrates the architecture of a FCD concept developed at Mannesmann Autocom (Fastenrath; Friedman, 2008).

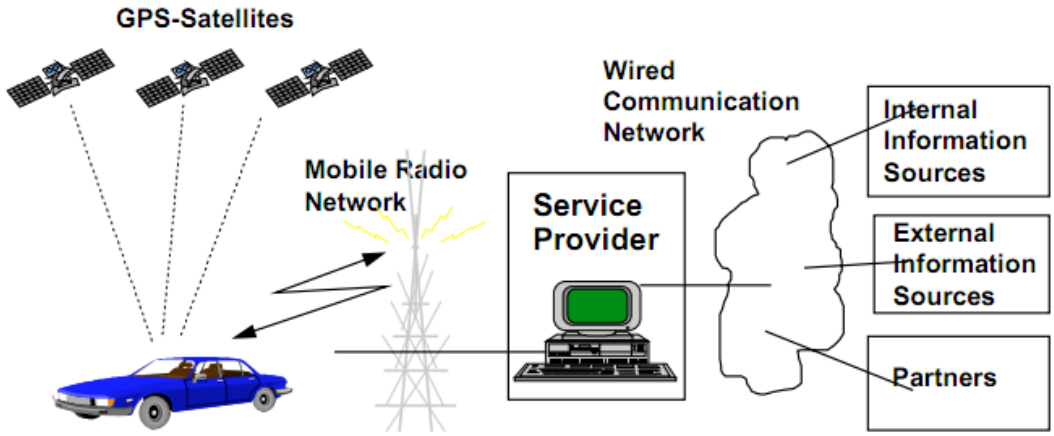


Figure 3 - Floating Car Data, courtesy of (Fastenrath)

Data collected through FCD create more transparency on transportation and therefore various FCD techniques have been researched and developed to improve the efficiency of urban transportation (Turksma, 2000).

1.2.2. Applications

FCD is a promising mean to improve individual travel services as well as public traffic management. The applications of FCD can be categorized into five categories (Ahmadi, 2010):

Transportation planning: the real-time data provided by FCD is a good means for calibration of traffic models. Furthermore, it offers invaluable source of information for policy making and infrastructure investment decisions in cities with high volume of freight transport.

Traffic monitoring and planning: FCD provides real time traffic data of the network, which makes it possible to deduce the traffic situation and inform travelers about the average speed, congestion on the links and average travel times in contrast to conventional methods that are based on traffic data collectors like loop detectors. Obtaining real-time traffic information is invaluable since it provides a clear view of traffic situations in a particular area (Koutsopoulos, et al., 2010). As an example Koutsopoulos et al used FCD in Stockholm to monitor the travel time between two particular points in the city during different hours of a day. The results proved to be of tremendous potential for predictions of traffic patterns during those hours between those particular points for successive periods.

Traffic management and network performance monitoring: digital road maps are necessary for vehicle navigations, route guidance and traffic simulation. However such maps are not available for all parts of the world. Furthermore, keeping road maps updated to reflect the ongoing changes in the network has rendered a challenging task. Using FCD generated from taxis the real-time skeleton of the road map of the city can be reconstructed with sufficient coverage.

Figure 4 is the comparison of an FCD generated road map skeleton to NAVTEQ road network of the same area (Ahmadi, 2010).

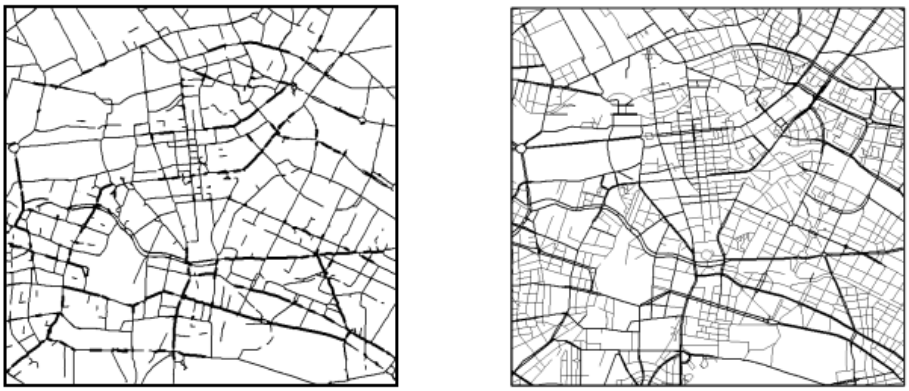


Figure 4 - FCD generated road map skeleton vs. NAVTEQ road map

An example of application of FCD in routing strategies is feeding the data to the Variable Message Signs on the road network to show reliable travel-time on alternative routes. Another application is in logistic companies that have to schedule a number of trips per day to deliver goods. FCD can be used to refine the scheduling strategies traditionally based on historical travel-times and influence the routes and sequence of deliveries.

Environmental and personal applications: commercial companies like Tom Tom and Garmin use FCD data for optimized route planning which is then offered as a service the users of their devices. Other applications of FCD in this area include usage base vehicle insurance which is on its way to replace the traditional systems for calculating insurance premium based on vehicle type, driver's gender and age.

Other applications: governments use FCD to adjust the toll fees on the roads based in order to connect the driving distance to the fee.

1.3. Objectives

Despite the mentioned applications, little research has been done for applying visualization techniques to the floating car data. This thesis looks at the role of information visualization for deducing information from the collected and processed data that has not been feasible by other means. In other words, it adds a new dimension in by using presentational elements to enrich the deduced data and creates a new grasp to the result of FCD data processing.

The goal of this thesis is to design a visualization framework that can visualize geographical and traffic data. Extensibility is of major focus in this work so that other sources of data can be fed to the framework and be visualized on road maps.

There are a variety of visualization scenarios that can be considered for the FCD accessible to this work. The following scenarios are the ones selected for this thesis:

Real-time FCD visualization: the FCD received describes the state of the network in a sequence of moments. This data is used to animate the road network and vehicle positions in a real-time fashion with the capabilities to track individual vehicle along their journey.

City travel planning: dynamic route planning in the Stockholm area enables the user to receive route suggestions for two arbitrary locations in the city based on the real-time data received from the taxis.

It's worth mentioning that as further work on this thesis, other types of information derived from FCD can be fed to MapViz visualization framework and be visualized in the same fashion as the above scenarios.

The next chapter describes the sources and type of FCD received during this thesis and used for visualization.

2. Data

This chapter describes the type and sources of FCD received and used in this thesis for the visualization purposes.

2.1. GPS Probes

This data is collected from about 1500 taxis equipped with GPS devices traveling in the Stockholm area at the time of writing this thesis. The number of GPS probes read per minute exceeds 1000 readings per minute.

Taxis reports GPS probes based on either distance or time depending on their speed status. If a taxi is driving fast for example on a motorway, generation of probes tends to be more time based and they are reported every minute. In lower speed conditions, the probes are generated every 400 and 600 meters depending on the taxi being free or hired respectively. Despite the expected frequency of probe reports, the reported data in practice contains more gaps between successive probes. Figure 5 shows the distribution of gaps between successive probes (Rahmani et al., 2010).

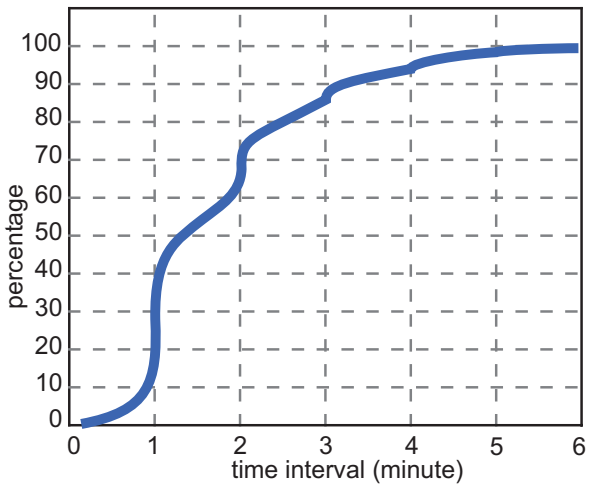


Figure 5 – Cumulative distribution of time gaps between pairs of consecutive probes, courtesy of (Rahmani et al., 2010)

The instant speed and heading of vehicle is not reported. That is partly because fleet dispatching systems do not need it, and also because of minimizing the cost of data transmission. The status of a taxi is represented as a binary field free/hired. Furthermore, the number of probes received varies depending on the number of active taxis logged in to the dispatching system. This often varies based on day of week and time of day. Collected data shows there is a higher demand for taxis during workdays in the morning and afternoon and also on Saturday nights. Figure 6 displays the distribution of probes received between 6:00 and 20:00 from 18th January 2010 until 8th March 2010.

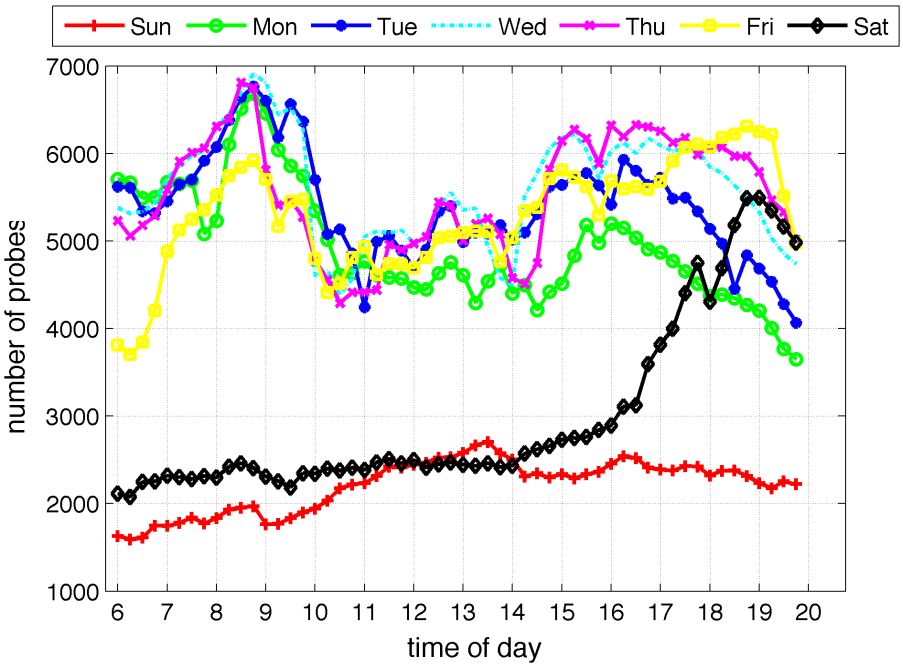


Figure 6 - Distribution of probes over weekdays (Rahmani et al., 2010)

Each reported probe from the taxis contains the following data elements:

- Vehicle ID
- Timestamp of the GPS probe
- GPS Position in form of latitude and longitude
- Occupied or free

MapViz doesn't consume this raw data directly and receives the result of map-matching process, which is described in the following chapters.

2.2. The Road Network

A road network is the digital model of a city modeled as a graph consisting a large number of nodes and links. Each link is specified with two nodes. Each link in the road network contains among others the following information:

- Unique link id
- Starting node id
- Ending node id
- Length
- Speed limit
- If the link is part of a roundabout
- If the link is part of a tunnel
- If there is a traffic light by the end of the link
- List of ids of the next links
- List of latitudes for the start, end and midpoints of the link
- List of longitudes for the start, end and midpoints of the link

NAVTEQ, a GIS provider company, has contributed the digital road network used in this thesis. Nevertheless, other road networks like NVDB[5] and OpenStreetMap can also be used and are supported.

The next chapter describes the development environment used for creating the MapViz platform. Considering the variety of tools and environments to choose from, each section describes the alternatives and the reasoning for the selected tool and environment.

[5] Nationell vägdatabas (Swedish national road database)

3. Development Environment

MapViz is a web-based platform and there are a variety of technological alternatives targeted for such applications. This chapter describes the technologies, tools and environments used for creating the MapViz platform and the rationale behind each choice.

There are many programming languages available each with advantages and disadvantages and often several languages suite specific requirement of an application. In this thesis, Java is used as the main programming language due to the existence of a large user community, variety of open-source frameworks and development support tools and last but not least the writer's familiarity with this programming language. Besides a programming language, a development environment consists of several other elements that are discussed and reviewed in this chapter.

3.1. Integrated Development Environment

IDEs are comprehensive development applications that provide tools for writing, compiling, editing and debugging code. Although there is no standard on the exact tools provided in IDEs, the mentioned functionality is the minimum available in most modern IDEs. In addition to the tools above, some IDEs provide assistance for build management, version control, GUI development, etc. IDEs goal is to enhance productivity and increase the development velocity in projects.

There are multiple commercial and free IDEs available for Java language. Most popular IDEs on the market at the time this thesis is written are:

Eclipse: a free and open-source multi-language IDE

JBuilder: a commercial IDE by Embarcadero Technologies

NetBeans: a free multi-language IDE provided by Oracle. NetBeans is a reference IDE for the Java language.

IntelliJ IDEA: a commercial multi-language IDE by JetBrains. A community version of IntelliJ IDEA with limited features is available for free.

JDeveloper: a free IDE provided by Oracle

Although IntelliJ IDEA is considered to be one of the smartest IDEs on the market, due to license limitations Eclipse has been chosen for this project. Eclipse has a large user-base, which simplifies finding community knowledge about the IDE. There is also a remarkable number of plug-ins available for Eclipse in order to add functionalities to the IDE that are not provided by default.

3.2. Version Control

Version or source control is an aspect of Software Configuration Management (SCM) is referred to the management of changes to versions of programs, documents and files. Version control systems make it possible to track changes or go back to previous versions of project files. The need for version control systems is most apparent when multiple people modify the same file and people own different versions of the same file. Several free and commercial version control systems are available on the market. The most popular source versions are:

Subversion (SVN): a free open-source centralized version control system. Subversion is part of Apache Incubator and is recognized as one of leaders of standalone version control systems(Schwabe, 2007).

Concurrent Version System (CVS): a free open-source centralized version control system

Git: a distributed source control system developed by Linus Torvals for Linux kernel development

Mercurial: similar to Git with focus on performance and scalability

StarTeam: a commercial source control system by Borland

Subversion is chosen for this project because of access to free repository providers and good support in Eclipse IDE.

3.3. Build Tools

Build tools automate a set of day-to-day development tasks like compiling code, packaging binaries, running tests, deployment and creating documentations. There two popular build tools available for Java projects:

Ant: an xml based build tool and a member of Apache projects

Maven: an xml based project management and build tool and a member of Apache projects. Maven can also manage dependencies between projects.

Maven is chosen for this project due to its capabilities in dependency management, which is missing in Ant.

3.4. Application Server

An application server is middleware software that provides an environment where applications can run. The general difference in application servers is in the services they provide to the applications running on top of them. Since MapViz is built using Java EE[6] technologies, an application server capable of providing Java EE services required for proper execution of MapViz. There are several popular open-source application and web servers that qualify for MapViz:

Jetty: a simple http-server and servlet container by Eclipse Foundation

Tomcat: a Servlet container that supports parts of Java EE by default and is capable of providing other services through add-ons. Tomcat is provided by Apache Software Foundation.

Resin: an application server available both as commercial and open-source versions by Caucho.

JBoss: an application server supporting full Java EE specifications available both as commercial and open-source by Red Hat.

Glassfish: an open-source application server supporting full Java EE specifications provided by Oracle as a Java EE reference application server

Due to popularity, lightweightness and large user-base, Tomcat is used for development of MapViz. Nevertheless, due to using standard technologies during development of MapViz, it is possible to run it on other application servers with minor modifications.

[6] Java Enterprise Edition

3.5. Testing

Testing is an essential phase of software development to ensure the quality of the product. Test automation is an important step toward software quality by enabling frequent execution of tests without further overhead on the development process. JUnit, a Java framework for unit testing, is used in this project to produce unit tests, which are executed as a part of build process by Maven.

In order to build a framework, there are fundamental choices that need to be made which has profound impact on the result and undoing them often requires nontrivial amount of work. The first choice to be made in this thesis is the GeoData format used for encoding the data discussed about. The visualization process is deeply bound to the selected format and it determines how the data should be interpreted. The available choices and the reason for the selected format is discussed in section 4.1.

Visualization of FCD takes place on top of a map. Creating a map from scratch takes substantial amount of work and is out of the scope of this thesis. Therefore a map provider service is needed in order to be able to visualize FCD on top of that. The alternatives considered and the selected solution is discusses in section 4.2. Furthermore, users and machines often speak a different language when it comes to geographical data. Translation between the user and machine language is done through geocoding services which is discussed in detail in section 4.3.

The rest of the next chapter details the architectural decision made along the development of MapViz platform and elaborate on the internal design of the platform.

4. Design and Implementation

MapViz is a request intensive application that requires special architectural considerations. The challenges encountered during the design phase of MapViz and the choices made are reviewed and discussed in this chapter.

4.1. GeoData Formats

In order to visualize geographical data, it needs to be encoded in a consumable format. This chapter describes the GeoData format choices available and the reasoning behind the selected format in this thesis.

GeoData formats are standard ways to encode geographical data into a digestible form for machines. Various agencies (e.g. Open Geospatial Consortium[7]) and companies (e.g. Keyhole[8] and Google) have created GeoData formats for specific purposes, which entail advantages and disadvantages for particular use cases. Geographical data often includes coordination system description, elevation data and shape layers for expressing drawings, streets and postal zone boundaries.

GeoData formats are generally divided to two categories: Vector and Raster (GIS Cookbook, 2007). Vector is referred to use of geometrical primitives such as points, lines and shapes to represent geographical data. Raster is referred to image

[7] OGC is an international standard organization consist of commercial, governmental, non-profit and research organizations with the goal of encouraging development of open standards for geospatial content and services

[8] Keyhole was acquired by Google in 2004

information stored as a matrix of pixels. Raster formats are generally appropriate for representation of continues dense data e.g. elevation and have a limited resolution determined by the pixel size. Vector formats on the hand best suite sparse data like particular geographical locations and have no resolution limits. Considering the structure of floating car data, vector formats are a better fit for this project. Representation of reality using these two alternatives is explained in Figure 7.

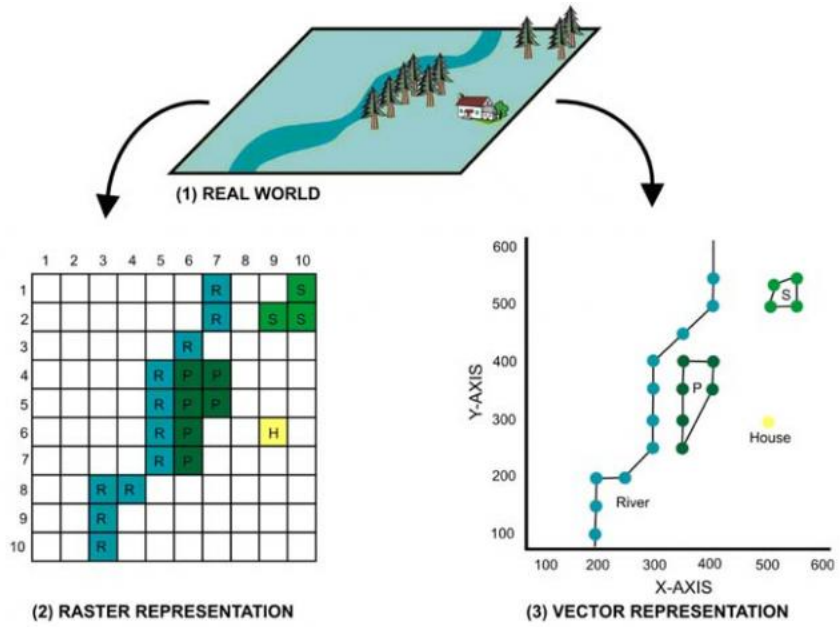


Figure 7 - Comparison of Raster and Vector Models, courtesy of (GIS Cookbook, 2007)

Vector formats are the most common form of GeoData representation and are used in many GIS[9] applications. There are several GeoData formats available in the GIS ecosystem sponsored and developed by commercial and non-commercial actors. Variation of data formats leads to inability of geographical systems to communicate with each other unless the data is converted to the target format. This intrinsically limits the interoperability between systems and adds an overhead of developing conversion tools for each specific data format. Furthermore, some of the formats are proprietary, which restrains their applications to certain parties.

Problems mentioned above, led to a movement towards open standards in order to foster interoperability between systems and reduce the overhead of data conversion. Some of the most common standard vector formats are described below.

4.1.1. Geography Markup Language (GML)

GML is the XML[10] grammar defined by the Open Geospatial Consortium (OGC) to model geographical data(GML, 2007). GML is designed to serve as a standard modeling language for geographical systems as well as an open interchange format for geographical data exchange among applications or over Internet. The purpose of

[9] Geographic Information System
[10] Extensible Markup Language

GML is to increase interpretability and reduce costly geographic data conversions between different systems.

World is modeled in GML as a set of feature. Feature is an abstraction of a real world phenomenon (GML, 2007) e.g. a building. Each feature has a state that is described by a set of triple properties (name, type and value) e.g. position of the building.

Figure 8 is the representation of polygon, point and line in a GML document.

```
<gml:Polygon>
    <gml:outerBoundaryIs>
            <gml:LinearRing>
                    <gml:posList>0,0 100,0 100,100 0,100 0,0</gml:posList>
            </gml:LinearRing>
   </gml:outerBoundaryIs>
</gml:Polygon>
<gml:Point>
   <gml:posList>100,200</gml:posList>
</gml:Point>
<gml:LineString>
   <gml:posList>100,200 150,300</gml:posList>
</gml:LineString>
```

Figure 8 - Sample GML Document

GML is a language to encode geographic content for any application by describing objects and their properties. However GML doesn't include the visualization of geographic content. In order to visualize GML information, other technologies are required to transform the data to some graphical form. This is commonly achieved by transforming GML to SVG[11] using XSLT[12], which can be rendered and displayed by browsers.

4.1.2. Keyhole Markup Language (KML)

KML is an XML-based notation for expressing and visualizing geographic data. KML was designed for a product called Keyhole Earth Viewer by Keyhole, Inc, which was acquired by Google in 2004. Keyhole Earth Viewer was later renamed to what is known today as Google Earth (KML). KML was approved by Open Geospatial Consortium in 2008 and became an international open standard(Shankland, 2008).

KML shares some of the same structural grammar as GML and like GML specifies a set of features. Figure 9 displays a KML document describing a point on the New York City.

One important difference between GML and KML is that KML is capable of encoding visualization information while GML is a pure content representation and relies on other technologies for visualization.

[11] Scalable Vector Graphics (SVG) is a family of specifications used to describe two-dimensional vector graphics
[12] Extensible Stylesheet Language Transformations (XSLT) is an XML-based language used to transform XML documents

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
<Placemark>
  <name>New York City</name>
  <description>New York City</description>
  <Point>
    <coordinates>-74.006393,40.714172,0</coordinates>
  </Point>
</Placemark>
</Document>
</kml>
```

Figure 9 - Sample KML Document

4.1.3. Shapefile

Esri shapefile is a mostly open specification for expressing geographic data. It is developed and regulated by a company called Esri. Shapefile is capable of modeling geometries i.e. points, lines and polygons and their attributes.

Although shapefile is a common expression, it in fact refers to a set of several files expressing the geometry shapes, what they represent, indices to improve performance, etc. The actual shapefile refers to files with ".shp" extension however they alone are incomplete for distribution and the supporting files are required.

Shapefile is a binary format and cannot be read or written without special programs. Currently, there are several free and non-free programs available on the market for reading and writing shapefiles.

4.1.4. GeoJSON

GeoJSON is an open format based on JSON[13] notation for encoding a variety of geographic data structures (Butler et al., 2008). GeoJSON allows geographic data to be stored in a human-readable fashion. Figure 10 shows a GeoJSON document describing a point.

The advantage of GeoJSON is that it is more compact than XML, which makes it suitable for web application environment. However it can only express geographical content and relies on other technologies for visualization.

4.1.5. Why KML?

There are several criteria to consider when choosing a geodata format. A list of important measures of this project follows:

Open Standard: publicly available and not bound merely to single commercial entities

Expressing presentation: able to describe visualization of elements besides the geographical content

[13] JavaScript Object Notation

```
{
    "type":"Feature",
    "id":"OpenLayers.Feature.Vector_314",
    "properties":{},
    "geometry":{
        "type":"Point",
        "coordinates":[97.03125, 39.7265625]
    },
    "crs":{
        "type":"OGC",
        "properties":{
            "urn":"urn:ogc:def:crs:OGC:1.3:CRS84"
        }
    }
}
```


Figure 10 - Sample GeoJSON Document

Text-based: to simplify development and reduce the processing overhead for web applications

Widely-used: for interoperability between geographical systems i.e. existence of other system using the same format as well as being able to send and receive data from other systems without conversion

Tooling: availability of support tools for development based on the format

Among the geodata formats reviews, KML is the one format complying with all the criteria mentioned. It is an open standard by OGC, XML based and widely used as an exchange geodata format among geographic system especially on the Internet. Adaptation of KML format has gone far enough that some consider it as a de facto standard for expressing geographical data. Furthermore, KML enables applications to easily export data to Google Earth, a geographical information system and map freely available from Google.

4.2. Map Providers

MapViz is a visualization project on top of a map. Therefore a map provider is needed for MapViz to be able to visualize the discussed scenarios. There are several free and commercial map providers available with proprietary API for third-party applications to be able to design applications on top of their services. The following map providers are considered for this project:

Google Maps: free up to 25000[14] maps per day and commercial afterwards provided by Google

Bing Maps: free up to 125000 sessions or 500000 transactions per year and commercial afterwards provided by Microsoft

[14] The usage limits are based on the terms of services of various providers at the time this thesis is written and they are subject to change.

Yahoo Maps: free for low-traffic applications and commercial afterwards provided by Yahoo

OpenStreetMap: free and editable collaborative map maintained by the community

From the user-experience perspective, Google provider is the best alternative. On the other hand considering the usage limits makes OpenStreetMap is the most viable option.

Choosing a map provider has a profound effect on the application since it binds the application to that specific provider. This is especially a risk since most of the providers are commercial companies and their usage terms are subject to change. Changing from a provider to another is a non-trivial task and is often called vendor lock-in in the industry.

What is required for MapViz to function properly is a map provider however no map provider in particular. Therefore MapViz is designed to be able to function on top of all mentioned map providers with extension points for emerging providers. This is achieved through using an abstraction layer between MapViz functionality and the functionality needed from the map providers.

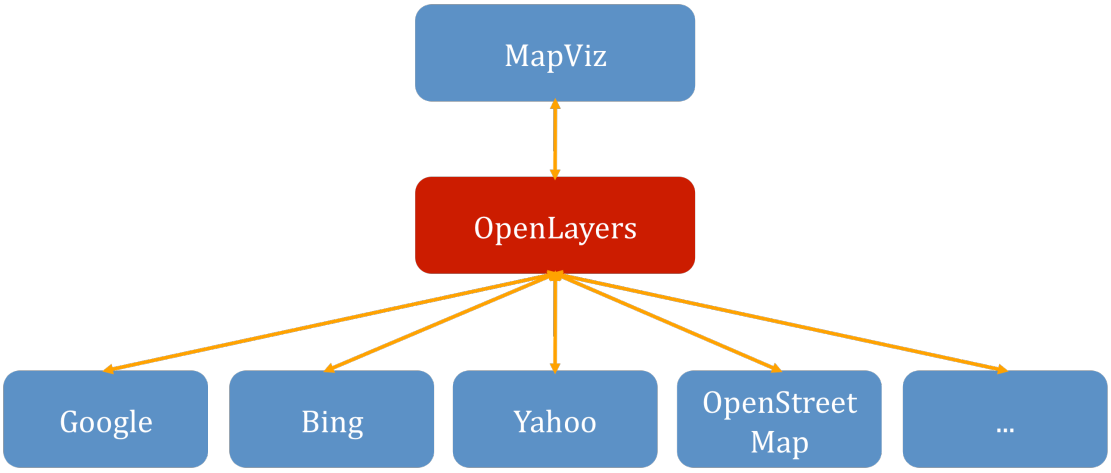


Figure 11 - OpenLayers

OpenLayers is an open-source Javascript library for displaying dynamic map data in web browsers. As displayed in Figure 11, it provides an API for communicating with the underlying map providers without tightly coupling the application with the map provider.

OpenLayers is sponsored by Open Source Geospatial Foundation (OSGeo), a non-profit organization that encourages and supports development of collaborative community-driven geospatial technologies and data (OpenLayers, 2011).

Using OpenLayers enables MapViz to defer the question of map provider to the runtime rather than the development time. Therefore, the choice of map provider becomes a matter of configuration and can be changed at any time without any effects on the application. This also eliminates the risk of updates in terms of

services and gives the power of choice back to MapViz to switch to another provider rather than being the victim of the change.

4.3. Geocoding

Geocoding is the process of finding geographical coordinates (longitude and latitude) from other geographical data such as street address and zip codes. Reverse geocoding is the reverse of this process, finding the associated textual location based on a give geographical longitude and latitude.

Figure 12 - Geocoding and reverse geocoding example

While geographical coordinates are conceivable almost by every geographic information system, they are not the optimal means of communication with end users. Users in general refer to geographical locations by names like street or city names. Therefore, geocoding and reverse geocoding is required in user facing applications to be able to communicate to user in their own language. Figure 13 demonstrates the role of geocoding and reverse geocoding in user facing applications.

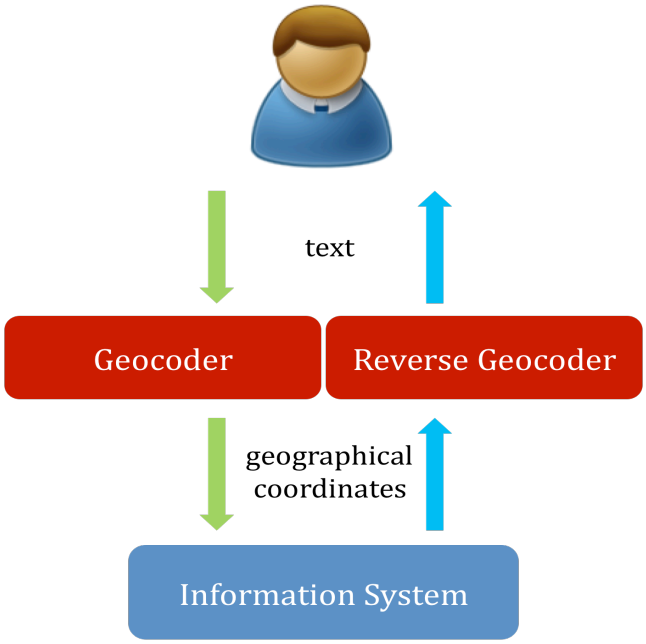


Figure 13 - Geocoding and Reverse Geocoding

Geocoder is a piece of software that performs the geocoding process. MapViz needs a geocoder in order to be able to communicate with the users in textual forms. Details of how geocoders function are beyond the scope of this thesis and a third-party geocoder is used in this project.

There are many free and commercial geocoding providers available in the market. Most map providers (all the ones described in the previous section) also provide geocoding services. Choice of geocoding service like the map provider bounds the application to certain providers. Most of the free providers have certain limit on the number of queries per day and service becomes commercial beyond that limit.

Following the same pattern as for the map providers, MapViz solves the problem of tight coupling by deferring the choice to runtime and create a neutral cross-provider solution. This is achieved through creating an abstract layer that encapsulates the functionality of geocoding and reverse geocoding. This layer is called the *geocoding layer* in the rest of this document. The geocoding layer delegates the geocoding requests to one of the available geocoding providers and hides the details of specific providers from the application. This approach minimizes the effort of switching between geocoding providers and limits the changes required to the abstraction layer.

MapViz in fact takes this approach one step further and creates a plug-in system in the geocoding layer that accepts several geocoding providers as the same time. For each geocoding request sent to the geocoding layer, the highest priority geocoding provider is used. Should the number of requests exceed the number of allowed request or any other failure on the providers' side, the geocoding layers disables that particular provider for a configurable period of time (24 hours configured by default) and switches to the next geocoding provider based on the priorities. This hierarchical request processing is continued until geocoding layer succeeds in getting a successful response from one of the providers. Once the suspension period has passed, the geocoding layer automatically enables the provider and adds it to the list of registered geocoding providers.

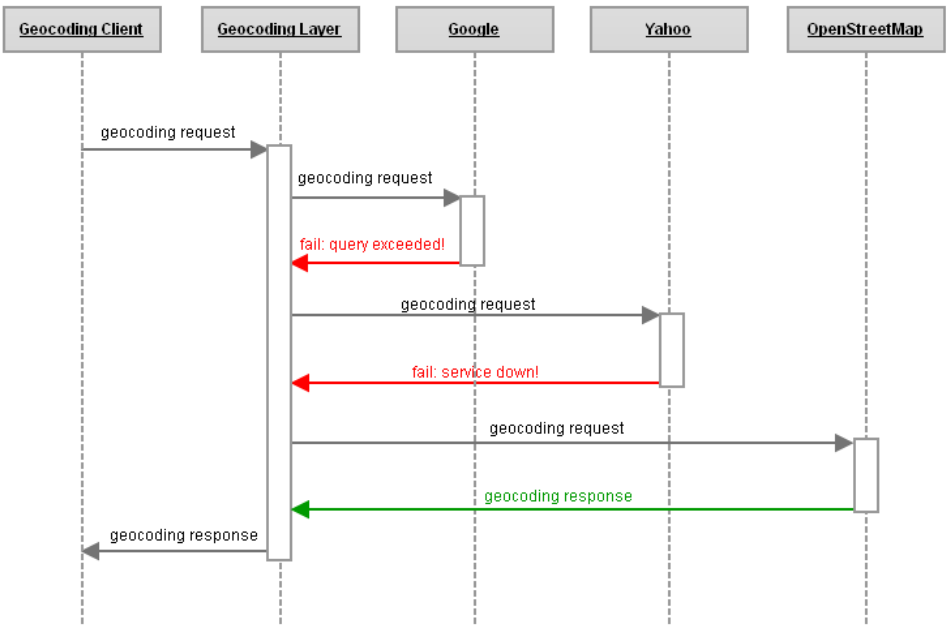


Figure 14 - Geocoding Layer Sequence Diagram

There is no limitation in the number of geocoding providers that can be plugged into the geocoding layer. In the default implementation of the geocoding layer, a plug-in for Google geocoding API is created as a proof of concept. The class diagram of the geocoding layer is available in Appendix B.

Figure 14 demonstrates a sample scenario in the geocoding process:

1. Client (upper layers of the application) sends a geocoding request containing the text to be geocoded to the geocoding layer.

2. The geocoding layer looks for the geocoding provider with the highest priority among the registered providers. It finds Google and sends a geocoding request to Google through its geocoding API.

3. Google returns an error response stating that the number of geocoding requests sent from MapViz have exceeded the permitted daily limit and would not accept any more requests for 24 hours.

4. The geocoding layer disables Google in the list of providers for 24 hours and looks for an active provider with the highest priority.

5. The geocoding layer finds Yahoo provider and delegates the request to Yahoo geocoding API.

6. Yahoo geocoding API responds with an error stating that the service is temporarily down.

7. The geocoding layer disables Yahoo for 24 hours and looks for the next active provider with highest priority.

8. It finds OpenStreetMap and sends the geocoding request to OpenStreetMap geocoding API.

9. The geocoding layer receives a successful response from OpenStreeMap and returns it in its own turn to the client.

4.4. KML APIs

As discussed in previous chapters, KML is used in MapViz as the geodata format. OpenLayers is capable of parsing and displaying KML documents. In order for MapViz to produce KML documents, a Java library is required to facilitate generation of KML documents based on geographical data. Several alternatives are available which are discussed below.

4.4.1. JAK: Java API for KML[15]

JAK is a free open-source library that provides an easy interface for accessing KML documents. JAK supports full specification of KML in addition to Google extensions to the markup language and uses JAXB[16] API to provide an object-oriented interface to KML.

The advantage of JAK is in its simple API, which is quite easy to understand and work with. Besides, it's a ready library and saves time in development by eliminating the work for creating a new library for the same purpose. The disadvantage of JAK is that it hasn't been active since Jan 2010 and doesn't support the latest Google extensions to the KML format.

4.4.2. Marshalling/Unmarshalling

This is a similar approach to what JAK provides. Marshalling is the act of converting Java objects to XML documents. Unmarshalling is the reverse of marshalling and converts XML documents into Java objects. Since KML is an XML format, unmarshalling can be used to create KML documents based on Java objects. There are several frameworks and libraries that are capable of marshalling and unmarshalling. JAXB (used also in JAK), XMLBeans and XStream are among the famous Java libraries for marshalling and unmarshalling.

The advantage of this approach is the simple Java friendly API and control over the parts of KML specification that is needed by MapViz. The disadvantage is the extra work needed to create the supporting classes for marshalling and unmarshalling.

4.4.3. Streaming API for XML (StAX)

StAX is a standard Java API for reading and writing XML documents. In contrast to other Java XML APIs like SAX[17] and DOM[18] it uses a cursor to traverse inside the XML document and pull information from the various parts of the document. StAX is a specification and needs an implementation to be able to be used in an application. Woodstox[19] is one of the popular implementation of StAX standard.
The advantage of StAX is full control over the parts of KML specification that is needed by MapViz. The disadvantage is that StAX is generally a difficult API to work with and in comparison to other approaches takes more time to write code that achieves the same result.

4.4.4. Why StAX?

MapViz is a request intensive web application. The more time spent on generation of KML documents, the less responsive the application will be and the slower the end user will perceive it. Therefore, it is important to make the KML generation as fast as possible. There are several criteria like simplicity of the API and required time for development that can be taken account when choosing one of the mentioned

[15] http://code.google.com/p/javaapiforkml/
[16] Java Architecture for XML Binding (JAXB) is a Java technology that allows XML documents to be mapped to Java objects and vice versa.
[17] Simple API for XML
[18] Document Object Model
[19] http://woodstox.codehaus.org

approaches. However the most important measure used in this project is efficiency and speed in order to ensure the shortest response time possible.

A benchmarking experiment was designed to assess the efficiency of the discussed approaches. Based on the results, StAX approach with Woodstox as the implementation is chosen for this project.

4.5. Architecture

As mentioned in the beginning of this chapter, MapViz is a request intensive web application. Number of concurrent users significantly increases the number of request sent to server. Scalability is the ability of an application to process a growing amount of work in a graceful manner. Therefore special care should be given to the scalability of the MapViz in order to be able to extend the number of request it can handle per second if needed. As demonstrated in Figure 15, MapViz is divided to two applications that can function independent of each other:

- MapViz Visualizer
- MapViz Data Services

MapViz Data Services is responsible for retrieve, transformation, and aggregation of floating car data. In other words, Data Services receives the raw FCD and prepares it for visualization. MapViz Visualizer is in charge of visualizing the transformed FCD and presenting it the user.

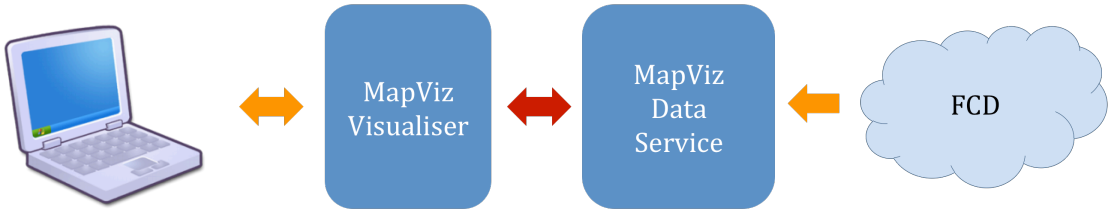


Figure 15 - MapViz Components

MapViz Visualizer and MapViz Data Services are self-contained standalone web applications and can operate independently as long as they can reach each other over the network. They communicate through a REST[20] interface and a well-defined JSON contract.

There are several advantages to the proposed architecture:

Reusability

Visualizer and Data Services are loosely coupled and communicate through a defined contract. This allows the components to be agnostic of each other and be able to feed or consume data from any other party that complies with the defined

[20] Representational State Transfer

contract. In other words, the Visualizer can be reused to visualize other sources of data and the Data Services can be reused to feed other types of visualizers. As an example, this architecture allows other visualizers for specific clients to be added to the system based on the data provided by the Data Services without implications on MapViz Visualizer. Such a usecase would a native iPhone visualizer application can be designed to consume the same data as MapViz Visualizer does and render the result on iPhones.

Same level flexibility is provided on the sources of data. The MapViz Visualizer is able to receive and visualize data from other sources e.g. real-time collective traffic information (bus and subway) or road cameras rather than MapViz Data Services without any implications on MapViz Data Services.

Figure 16 demonstrates a possible extension of MapViz with the real-time collective traffic information as an extra source of data and a native iPhone application as an extra visualizer.

Scalability

MapViz components are autonomous and can be deployed independently on separate application servers. This allows each component to scale separately as the number of request grows. There other scalability measures incorporated in each component's design that is discussed in later sections.

When new Visualizers (e.g. for iPhone) are added to the system, we expect and increase in the amount of load on the Data Service however it won't affect the amount of load on other visualizers (e.g. the web-based visualizer implemented in this thesis). In such a scenario, it will be required to add more resources to the Data Service and scale it up without affecting the Visualizers. The current design adds the required flexibility to handle such scenarios.

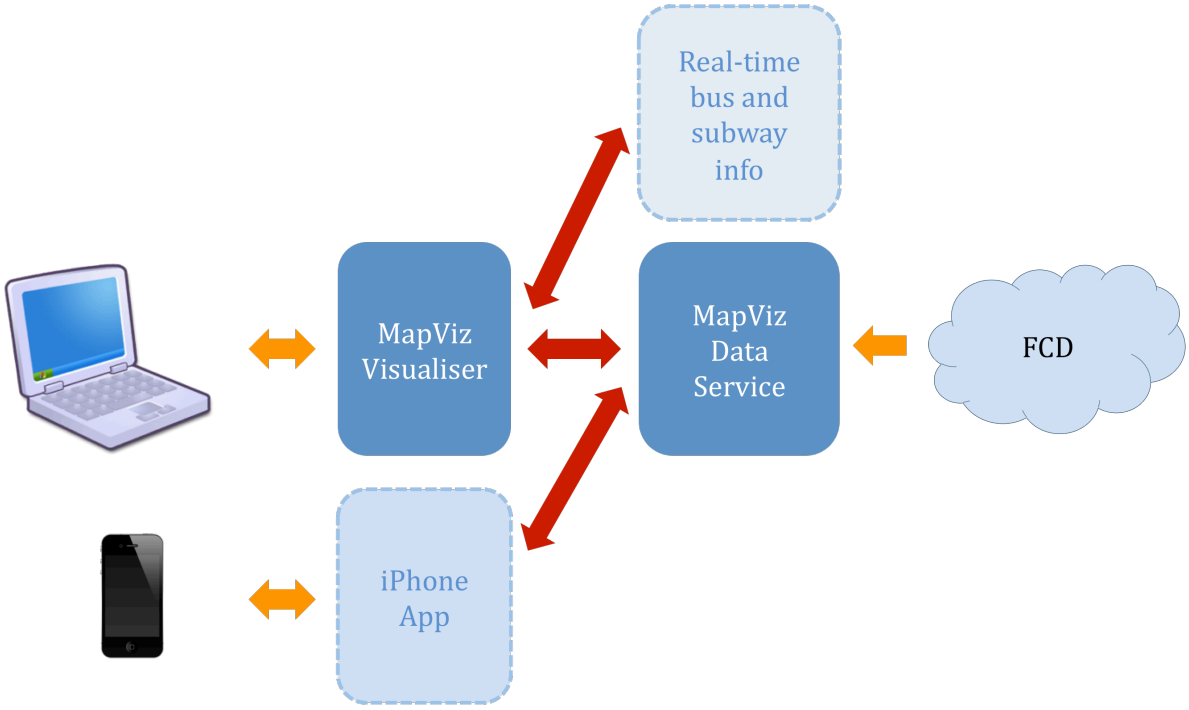


Figure 16 - Alternative data source and visualizers

Loose coupling

Coupling is the degree to which components of a system are dependent to each other (Coupling). A loosely coupled system refers to a system that its components have little or no knowledge of definition of each other (Stevens et al., 1974).

A tightly couples system has disadvantages such as a change in one component causes a ripple effect of changes in other components, time-consuming assembly of components due to the tight dependencies among them, decreased dependency since all the dependent components must be also included.

By separating the visualization functionality from data transformation and aggregation, MapViz components can operate independent of details of each other and decrease the coupling between components.

In the following sections, further details of each Visualizer and Data Services design are elaborated and discussed.

4.5.1. MapViz Data Services

The Data Services component is in charge of providing the required data for Visualizers. It is a web application heavily based on REST services taking advantage of the map-matching library delivered by Koutsopoulos and his colleagues (Koutsopoulos, et al., 2010).

Figure 17 demonstrates the architectural details of MapViz Data Services. The details of each component are described in the following sections.

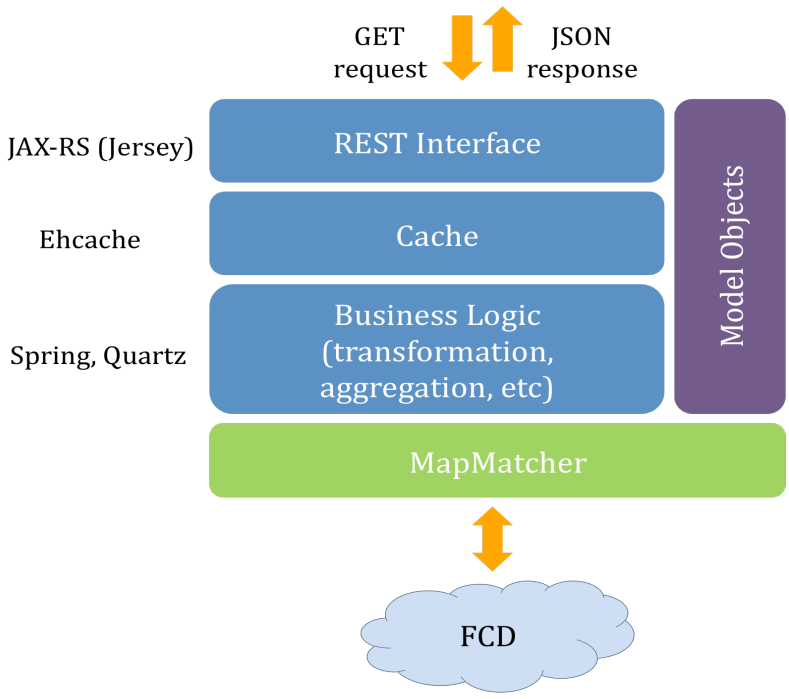


Figure 17 - MapViz Data Services Architecture

4.5.1.1. REST API

REST (Representational State Transfer) is a style of software architecture for distributed hypermedia systems such as World Wide Web (Fielding, 2000). In a simpler sense, it refers to any system that transfers data directly over HTTP[21] rather than additional layers e.g. SOAP[22] on top of HTTP. An architecture that complies with REST principles is called RESTful.

In a RESTful architecture, each resource is identified by a unique id, URI[23]. The client-server model in REST is based on the client requesting the representational state of a resource by its URI and receiving a response. There are several advantages to this approach:

Scalability: since all the state information required is encoded in the request, server doesn't need to maintain state between successive requests. This allows the application to easily scale out if needed.

Interoperability: REST relies on HTTP, which is a ubiquitous protocol. There are libraries available in most programming languages for interaction to HTTP protocol. This makes the services exposed through REST available to a wide range of clients without extra requirements.

Simplicity: REST is based on widely known standard technologies

REST goes way beyond what is explained in this section however further details of this architecture style is beyond the scope of this thesis.

The REST interface in MapViz defines a set of resources that can be retrieved through HTTP GET request. It takes advantage of JAX-RS[24] a standard Java EE[25] technology for building RESTful applications. Jersey[26] is the JAX-RS implementation used in MapViz Data Services. Table 1 to Table 4 elaborate the details of services provided by the REST interface including the parameters that can be sent to the services to tweak their operation, example request and example response received from the service. The specification of these services based on WADL[27] format is listed in Appendix A and available at runtime at the following url:

Data Services WADL: `http://mapvizhost/service/application.wadl`

Real-time Vehicle IDs

This service returns the list of all currently active vehicles. The returned response is a JSON array containing the vehicle ids.

[21] Hypertext Transfer Protocol
[22] Simple Object Access Protocol
[23] Uniform Resource Identifier
[24] Java API for RESTful Web Services
[25] Java Platform, Enterprise Edition
[26] http://jersey.java.net
[27] Web Application Description Language

URI	/vehicle/ids
Parameters	None
Sample Request	http://mapvizhost/service/vehicle/ids
Sample Response	[10002, 10004, 10005, 10006, 10007, 10009, 10010]

Table 1 - Real-time Vehicle IDs Service Info

Real-time Vehicle Positions

This service returns the current position of the requested vehicles within a geographical boundary. The returned response is a JSON object that maps the id of each vehicle to its geographical position (latitude, longitude).

URI	/vehicle/positions
Parameters	ids: A comma separated list of ids. If specified, the position of those vehicles will be returned. Otherwise, list of all active vehicles returned. bound: Left longitude, bottom latitude, right longitude and top latitude. If specified, the position of vehicles currently traveling in the specified area will be returned. Otherwise, the positions of all vehicles are returned.
Sample Request	http://mapvizhost/service/vehicle/positions?ids=10004,10009&accumulative=true&bound=18.0127719043,59.32497223552,18.2677680957,59.344394959536

Sample Response	<pre>{
	  "10009": {
	  "latitude": 59.32480898438048,
	  "longitude": 18.267660026071503
	  },
	  "10004": {
	  "latitude": 59.344474596716566,
	  "longitude": 18.074712269807538
	  }
	}</pre>

Table 2 - Real-time Vehicle Positions Service Info

Real-time Vehicle Trails

This service returns the trail of requested vehicles within a geographical boundary. The returned response is a JSON object that maps the id of each vehicle to the following information regarding that vehicle:

- Current position
- Current average speed
- List of positions that the vehicle has recently traveled

The list of recent positions is particularly invaluable for portraying the routes each vehicle has been traveling. This service is able to provide the recent positions in two modes:
Link: list of recent positions on the last link the vehicle is traveling. This is the default mode of this service.

Route: list of recent positions on the all links of the last route the vehicle is traveling. This mode is called accumulative.

URI	`/vehicle/trails`
Parameters	`ids`: a comma separated list of ids. If specified, the position of those vehicles will be returned. Otherwise, positions of all active vehicles are returned. `bound`: left longitude, bottom latitude, right longitude and top latitude. If specified, the position of vehicles currently traveling in the specified area will be returned. Otherwise, the positions of all vehicles are returned. `accumulative`: if `true`, the accumulative representation of vehicle track is returned. Otherwise only the link the vehicle is currently traveling on is returned. The default value is `false`.
Sample Request	`http://mapvizhost/service/vehicle/trails?ids=10004,10009&accumulative=true&bound=18.0127719043,59.32497223552,18.2677680957,59.344394959536`

Sample Response	{
	"10009": {
	"position": {
	"latitude": 59.30958872708277,
	"longitude": 18.131965635712348
	},
	"locations": [
	{
	"latitude": 59.3098,
	"longitude": 18.13269
	},
	{
	"latitude": 59.30958872708277,
	"longitude": 18.131965635712348
	}
	],
	"speedLevel": 4
	},
	
	"10004": {
	"position": {
	"latitude": 59.34069017581955,
	"longitude": 18.046621629455032
	},
	"locations": [
	{
	"latitude": 59.34063,
	"longitude": 18.04668
	},

	{
	"latitude": 59.34069017581955,
	"longitude": 18.046621629455032
	}
	],
	"speedLevel": 1
	}
	}

Table 3 - Real-time Vehicle Trails Service Info

Trip Planner

This service calculates the best possible routes between two geographical positions based on the real-time as well as historical floating car data of the traffic links connecting the positions.

The return object is a JSON array of routes containing the details of each suggested route.

URI	`/tripplanner/plan`
Parameters	`start`: longitude and latitude of the starting point `destination`: longitude and latitude of the destination point `time`: the time of the trip `starttime`: if `true`, the time parameter is considered the starting time of the trip. Otherwise it would be end time of the trip.
Sample Request	`http://mapvizhost/service/tripplanner/plan?start=18.0127719043,59.32497223552&destination=18.2677680957,59.344394959536&time=2011-12-13%2014:11&starttime=true`

Sample Response	{
	"start":{
	"latitude":59.30958872708277,
	"longitude":18.131965635712348
	},
	"destination":{
	"latitude":59.31958872708277,
	"longitude":18.141965635712348
	},
	"routes":[
	{
	"probability":0.89,
	"locations":[
	{
	"latitude":59.3098,
	"longitude":18.13269
	},
	{
	"latitude":59.30958872708277,
	"longitude":18.131965635712348
	}

<table>
<tr>
<td></td>
<td>
<pre>
            ]
        },
        {
            "probability":0.74,
            "locations":[
                {
                    "latitude":59.34063,
                    "longitude":18.04668
                },
                {
                    "latitude":59.34069017581955,
                    "longitude":18.046621629455032
                }
            ]
        }
    ]
}
</pre>
</td>
</tr>
</table>

Table 4 - Trip Planner Service Info

4.5.1.2. Cache

The cache component is responsible to cache the data for a certain period and reuse the prepared data on successive request instead of retrieving and recalculating the same data on each request. This component significantly increases the performance of MapViz Data Services by saving CPU cycles through processing the raw FCD data only once. Ehcache[28] is an open-source java library used in this component for caching purposes.

4.5.1.3. MapMatcher

The MapMatcher component is responsible for retrieving the real-time GPS probes reported by the taxis traveling in Stockholm and mapping them to the digital network of the city through a stream processing infrastructure. (Koutsopoulos, et al., 2010). This component is delivered as a Java library (jar file) and is one of dependencies of MapViz Data Services.

The MapMatcher uses the digital network of Stockholm and calculates the most likely paths taken by the vehicles based on the fixed locations reported as probes. The resulting data of map-matching process for a pair of probes follows:

- Distance of each probe to the starting point of the link it lies on
- Timestamp of the reported probes
- A sequence of links connection the to probes' locations
- Probability of the link sequence being the true path taken by the vehicle

It is worth mentioning that the details of how the map-matching process is hidden from MapViz and goes beyond the scope of this thesis.

[28] http://ehcache.org

4.5.1.4. Business Layer

This component is the place where the actual business knowledge of dealing with FCD is retained. When data is received from the MapMatcher, it can't be directly used to form a meaningful representation. It needs to be transformed, aggregated, enriched and refined to other meaningful forms of information that are closer to a presentational state and can be consumed by the Visualizer. Figure 18 demonstrates how data is processed and prepared for upper layers through the business layer.

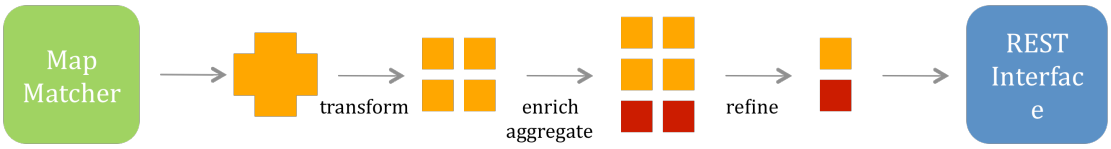


Figure 18 - Data Processing in the Business Layer

Target Timeframe vs. Relaxed Timeframe

Due to the infrequency of the probes received from the vehicles, sometimes the current position of the vehicle is not known. In other words, some vehicles might not have any current probe up to a few minutes, which makes it impossible to visualize their current state. To work around this limitation, the floating car data received in the project should be visualized with a certain delay. In other words, the visualization is in fact replay of the vehicle positions in a timeframe of from N minutes ago to the current time.

Furthermore, the MapMatcher used in this project does not include the routes that start or finish outside the requested timeframe. As demonstrated in Figure 19, consider the three probes received from vehicles A, B and C around the timeframe $[t_1,t_2]$. However, the outcome of map-matching process for the timeframe $[t_1,t_2]$ only includes vehicle B and discards the other two probes since they have started or ended outside the give timeframe. The result of this limitation is that the number of vehicles visible on the map in the beginning of period $[t_1,t_2]$ is quite small. This number gradually increases as it approaches the middle of timeframe and again diminishes as it closes the end of timeframe. This behavior is observed despite receiving almost an equal number of probes for the whole timeframe.

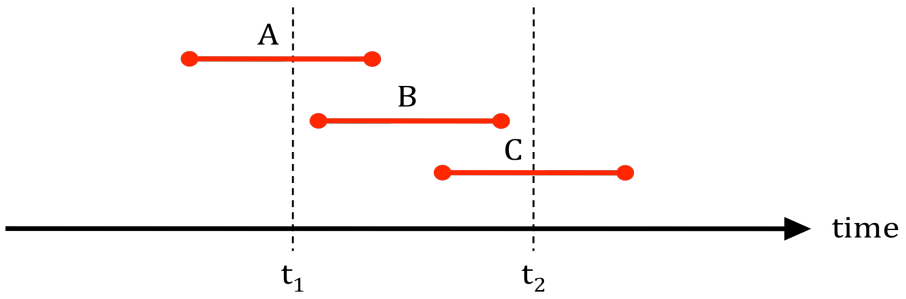


Figure 19 - MapMatcher Borderline Routes Problem

In order to get around the above issue combined with the involuntary visualization delay caused by sporadic probes, a concept of *Relaxed Timeframe (RT)* is introduced

and used in this project. For any *Target Timeframe (TT)*, the respective relaxed timeframe is defined as:

$$TT = [t_1, t_2]$$
$$\Delta = t_2 - t_1$$
$$RT = [t_1 - \Delta, t_2 + \Delta]$$

In order the receive all the routes received in the target timeframe of $[t_1,t_2]$, the relaxed timeframe of mentioned period is requested from the MapMatcher which is three times longer than the target timeframe. Since all three vehicle probes are within the relaxed timeframe, MapMatcher returns all three routes as the result of map matching. As a consequence of using the relaxed timeframe, there might be irrelevant probes included in the map-matching outcome such as vehicle D that is within the boundaries of the relaxed timeframe however outside the target timeframe. Such probes can be easily filtered out in a refining process in MapViz Data Services by comparing the target timeframe start and end time with of the probes.

Figure 20 demonstrates the relaxed timeframe for the target timeframe of above example.

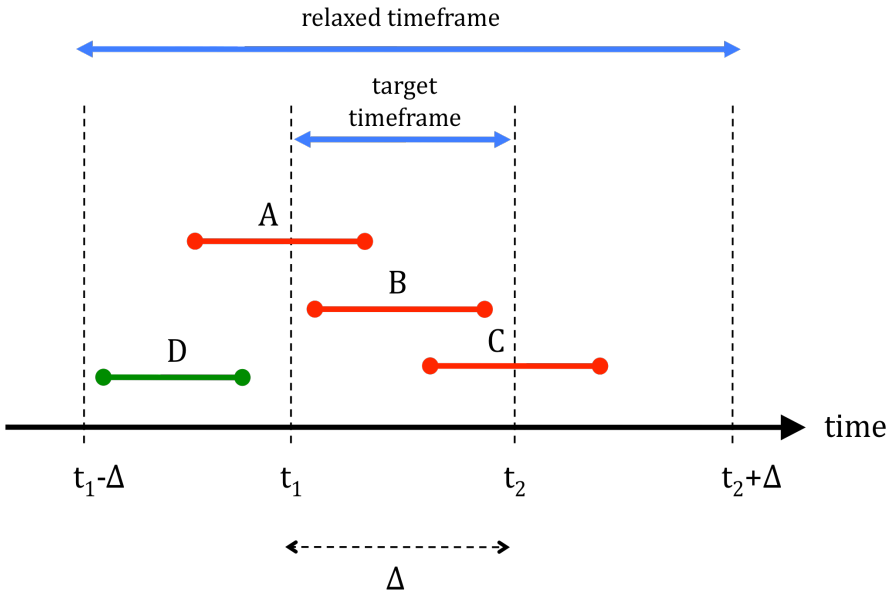


Figure 20 - Relaxed Timeframe vs. Target Timeframe

The Δ value used in MapViz is 5 minutes. In the current implantation, the relaxed timeframe always ends to the current moment. In other words:

$$t_2 = t_{now} - \Delta$$
$$t_1 = t_{now} - 2\Delta$$

In practice, this means the data is visualized by 2Δ delay (10 minutes). Thus, what is seen as the current position of the vehicles is where they used to be 10 minutes ago.

Position Calculation

The map-matching process for the request period results in a set of paths each comprised of a series of links. In order to visualize the movement of vehicles, the position of the vehicle needs to be calculated for each moment.

Figure 21 demonstrates an example of map-matching result. The target time representing the current moment shifted 2Δ back is represented by t_x.

In order to calculate the position of the vehicle at t_x, the first step is to determine if that moment lies on any of the paths available. That is done simply by comparing t_x to start and end time of each path. If no data is available for the vehicle at that moment, the vehicle is discarded from the map.

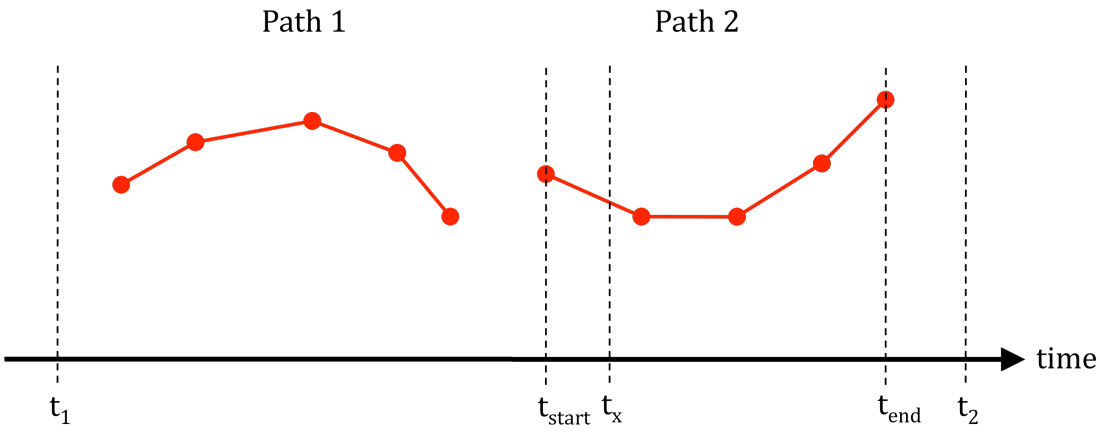


Figure 21 - Position Calculation

Assuming the data is available for t_x, the next step is to determine which link the vehicle has been traveling at that moment. Since the start and end time of each link is not available, the calculation is done based on the link lengths. The following formula is used to determine how far the vehicle has traveled on the path at time t_x:

$$l_x = \frac{(t_x - t_{start}) \times l_p}{t_{end} - t_{start}}$$

l_p = length of the matched path
t_{start} = time at the start node of matched path
t_{end} = time at the end node of matched path
l_x = distance the vehicle has traveled on the path at time t_x

The above formula is based on the assumption that the vehicle travels along the links with a constant speed and disregards the speed limit info available per link. A further work on this thesis would be to adjust the formula to take the speed information into account and project a closer-to-reality animation of the active vehicles.

Based on the distance traveled on the path and comparing it to the accumulative sum of link lengths on the path, the link the vehicle has been traveling on at t_x can be identified. To calculate the position on the link we need to calculate the bearing[29] θ between the link start and end points (Williams, 2011):

$$\Delta lon = lon_2 - lon_1$$
$$\theta = \arctan 2(\sin(\Delta lon)\cos(lat_2), \cos(lat_1)\sin(lat_2) - \sin(lat_1)\cos(lat_2)\cos(\Delta lon))$$

The position on the link can be calculated with the bearing value in the following formula:

$$lat_x = \arcsin(\sin(lat_1)\cos(\frac{d}{R}) + \cos(lat_1)\sin(\frac{d}{R})\cos(\theta)$$
$$lon_x = lon_1 + \arctan 2(\sin(\theta)\sin(\frac{d}{R})\cos(lat_1), \cos(\frac{d}{R}) - \sin(lat_1)\sin(lat_2))$$

In the above formula, θ is the bearing of the link in radius (clockwise from north) and d/R is the angular distance in radian where d is the distance on the link (from the link's start point) and R is the earth's radius, which is 6,371km (mean radius).

4.5.2. MapViz Visualizer

The Visualizer component is in charge of projecting the digested FCD received from the Data Services as an overlay layer on top of a map provider such as Google. Furthermore, it interacts with a geocoding provider in order to translate geographical positions to textual representation and vice versa.

Figure 22 demonstrates the architectural details of MapViz Visualizer. Spring and Spring MVC are the frameworks used for the business and presentation layers in MapViz Visualizer.

The details of the Visualizer component are described in the following sections.

4.5.2.1. Business Logic

The business layer of Visualizer component is responsible for retrieving the required data in order to be able to present the floating car data on a map. Geocoding abstraction as described in section 4.3 resides in this layer. The Geocoding layer interacts with a set of plug-ins for communicating with various geocoding services and picks an appropriate provider for each request.

[29] Bearing is measurement of direction between two points

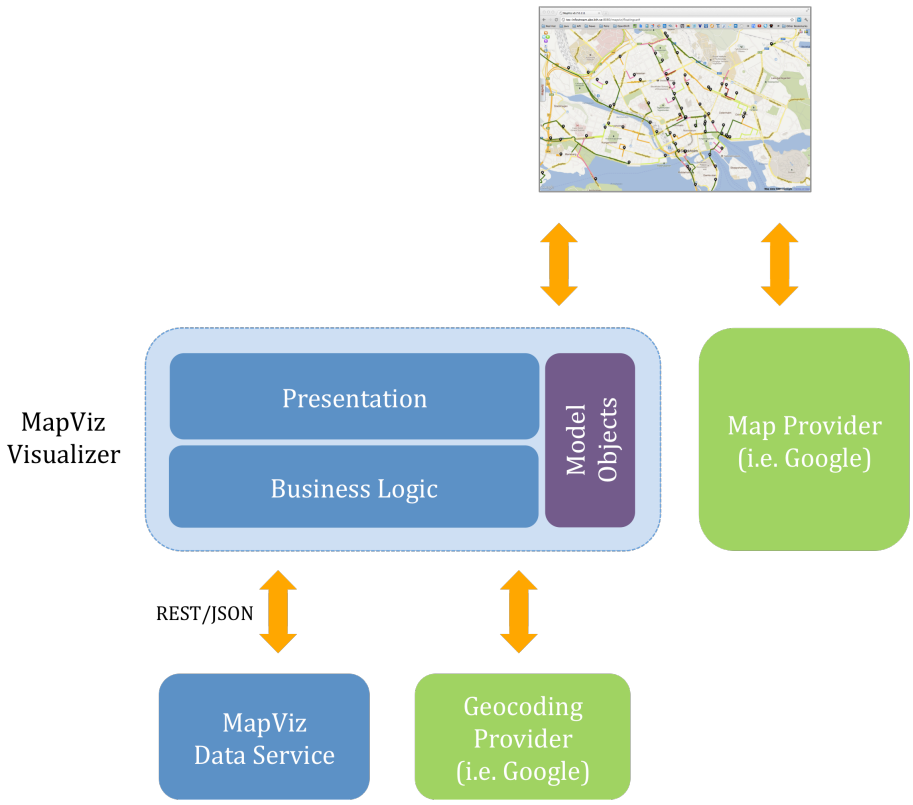


Figure 22 - MapViz Visualizer Architecture

The business layers does not directly interact with any sources of data out of the scope of Data Service and relies on the data aggregation happening in the Data Service. After retrieving the prepared data, the only processing happens in this layers is enriching visual elements of the data and making the final refinement for presentation. Color-coding and transformation to KML happens in this layer, which is fed later to OpenLayers for rendering.

4.5.2.2. Presentation

The presentation layer is responsible for projecting the transformed FCD on the map. This is done by taking advantage of the overlay layers in OpenLayers framework.

Using OpenLayers, the map data is retrieved from the map provider and displayed in the base layer. On top of that, there can be a number of overlays that are connected to various data sources and are positioned on the map. The overlays can contain geometric shapes like lines, polygons, etc and allow the user to interact with them. Despite being separate layers the result is perceived as one image by the user. Figure 23 demonstrates how overlays work.

The main window of the Visualizer can operate in two modes: control and full-screen. The control mode enables the user to modify the visualization parameters through a control panel and refine what is rendered on the map. The full-screen mode fills up the entire window of the map and is designed for one-way scenarios like display stands. Figure 24 and Figure 25 demonstrate the two operation modes of Visualizer.

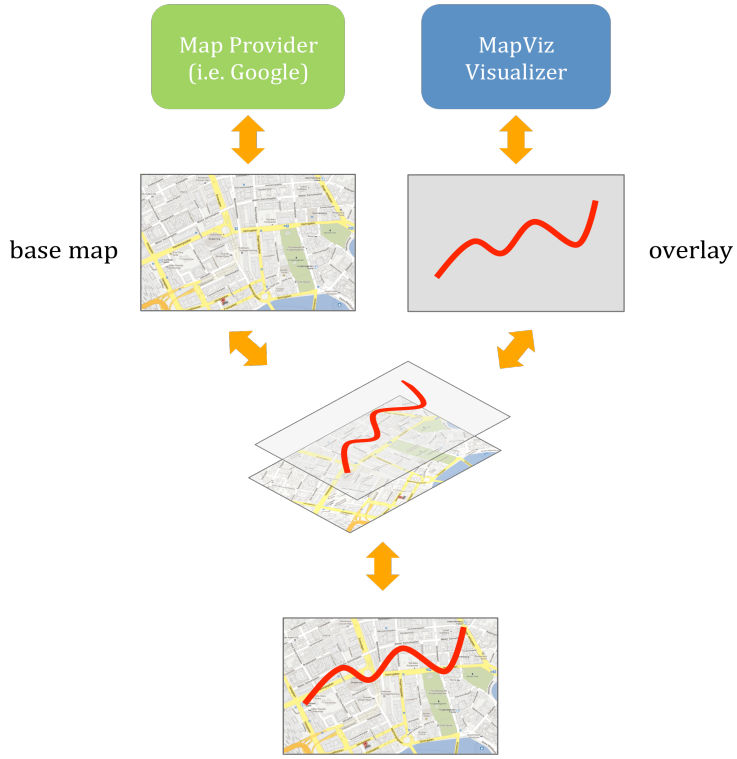


Figure 23 - Overlays

Real-time FCD visualization

The real-time FCD visualization is the real-time animation of the traveling taxi in the Stockholm city area. Taxis are displayed by a black icons pointing to the location of the vehicle on the map. The routes the taxis have been traveling can be presented in three ways:

No trail: in this mode no paths are rendered and only the current location of the vehicle is visible.

Single trail: this mode renders the current location of the vehicle as well as the link the vehicle is traveling on. When the vehicle continues to the next link, the previous link won't be highlighted and the new link will be marked instead.

Accumulative trail: in this mode the vehicle trails is highlighted from the beginning of the probe until the current location of the vehicle.

Figure 27, Figure 28 and Figure 29 demonstrate the three trail modes implemented in the Visualizer. Users can select or change the trail mode in real-time through the control panel.

Other control parameters provided to the user through the control panel are:

- List of active vehicle ids to display
- Display all active vehicles
- Pause and resume the animation
- Export the current state of the vehicles to KML file for opening in other applications like Google Earth
- Colorizing the vehicle trails to be able to distinguish overlapping trails
- Display vehicle ids

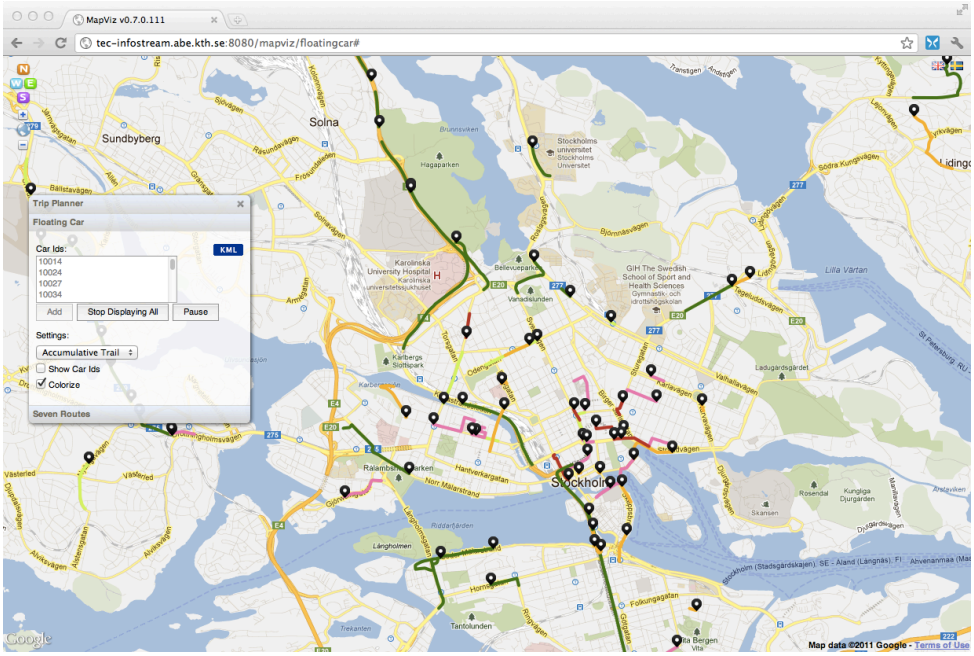


Figure 24 - MapViz Visualizer in control-mode

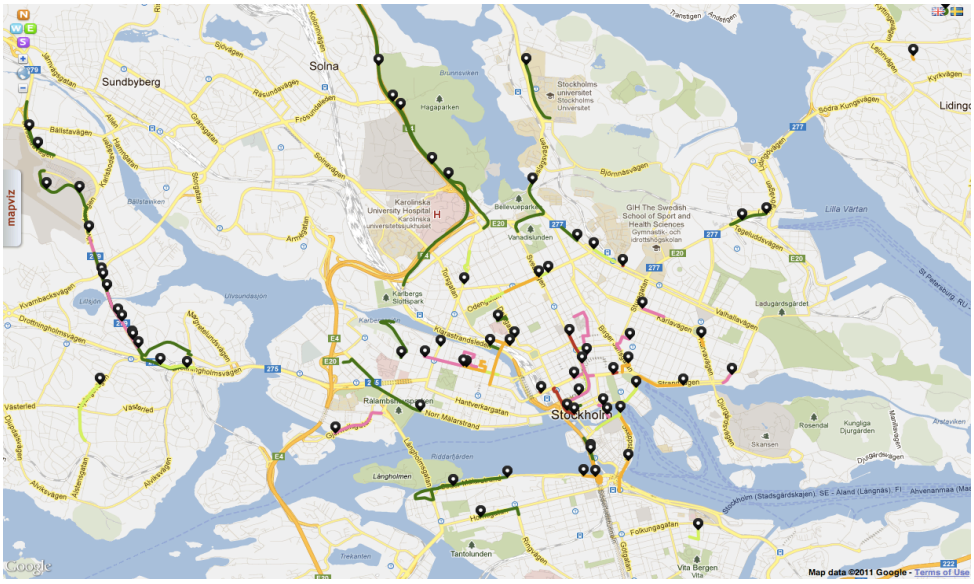


Figure 25 - MapViz Visualizer in full-screen mode

User can choose a set of vehicle ids to visualize particularly those vehicles on the map or click on “Display All” to display all active vehicles traveling in on the road network.

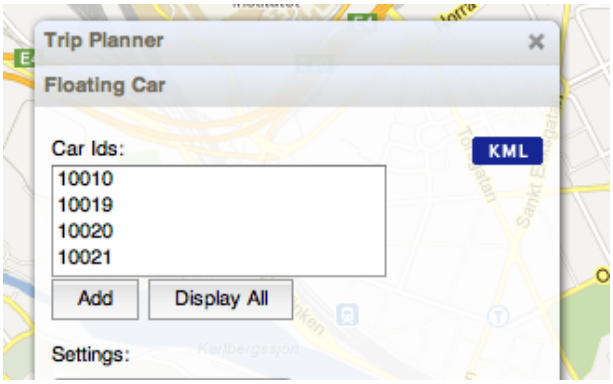


Figure 26 - Real-time FCD visualization export to KML

Using the “Pause” and “Resume” button, it’s possible to freeze the view of the vehicles in a certain state for example for taking a screenshot. The “KML” button allows exporting what is been displayed on the map to a KML file which can be later opened in application that support this format such as Google Earth. It’s recommended to pause the visualization before exporting to KML so that the vehicles are in a known state before exporting.

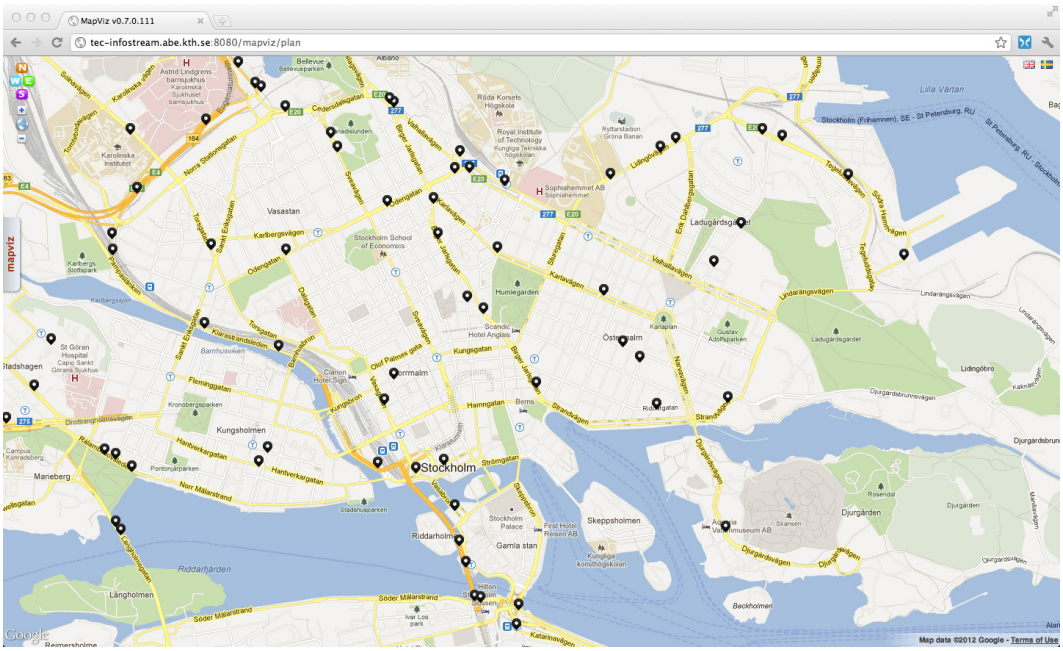


Figure 27 - Real-time FCD visualization in no-trail mode

Figure 30 shows a view of the real-time FCD with accumulating colorized trails and vehicle ids. The control panel is displayed on the left side of the screen and can transform to the full-screen mode by clicking on the cross (x) sign on the top-right corner of the control panel. This mode is particularly useful when the visualization is projected on a big screen for demonstration purposes. Clicking on the “MapViz” button on the left side of the screen in full-screen mode brings back the control panel to the previous state.

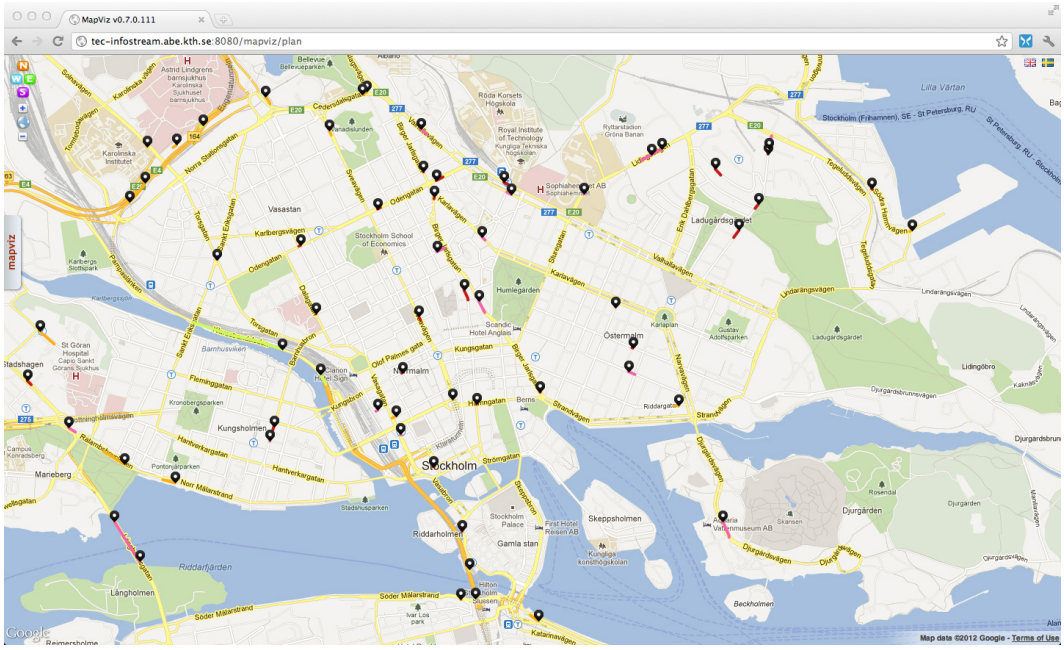


Figure 28 - Real-time FCD visualization in single-trail mode

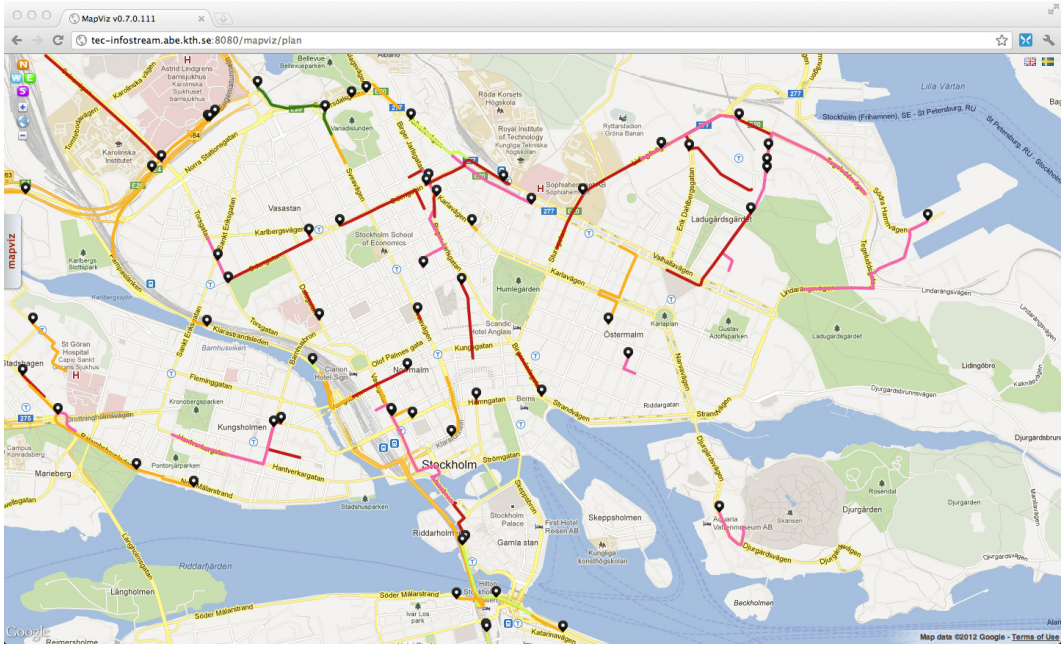


Figure 29 - Real-time FCD visualization in accumulative-trail mode

The smoothness of vehicle animation is highly dependent on how fast the vehicle data can be retrieved from the back-end. In order the minimize the data transfer hence increase the frequency of refresh, a boxing method is used to request the data strictly for the active vehicles visible in the current zoom window. With each request, the geographic coordinates (latitude and longitude) of the visible window in the browser are sent to the back-end so that the back-end while preparing the data for visualization filters out the vehicles traveling outside the visible window. Other factors affecting the animation's smoothness are the format of data used and

the efficiency of transformation of data to the target format as discussed in the above sections.

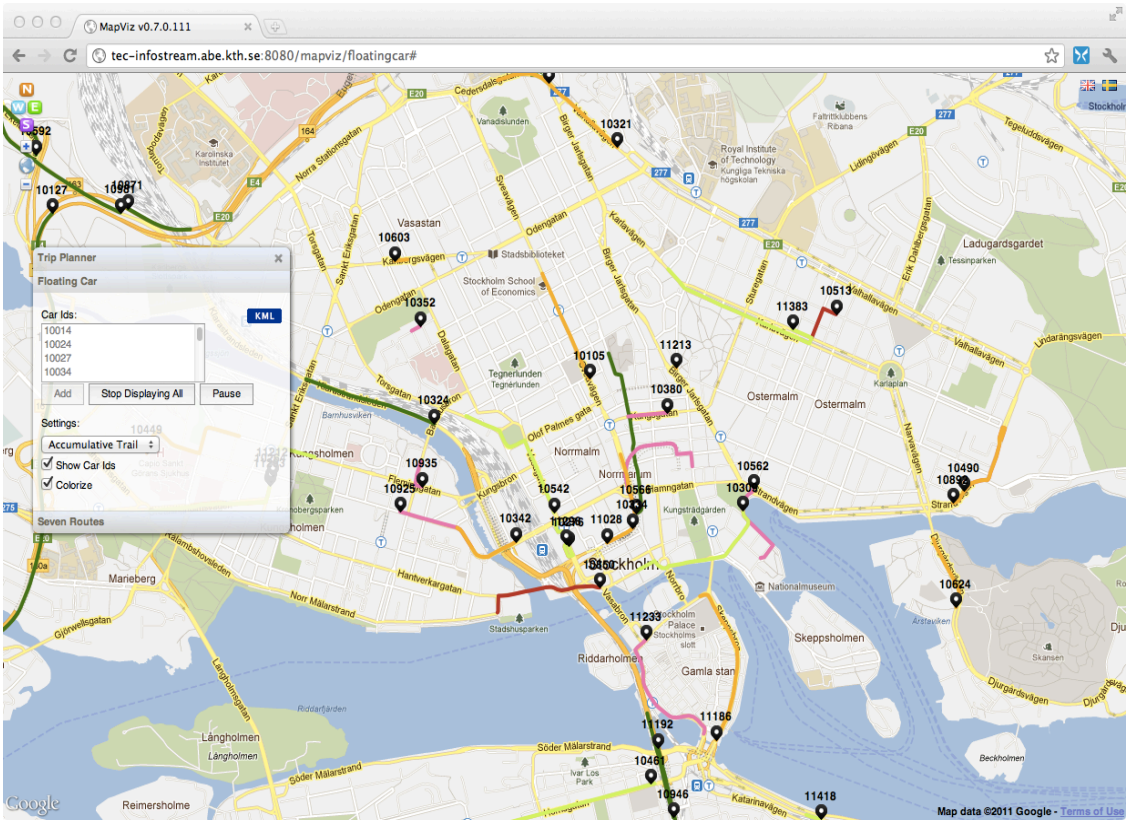


Figure 30 - Real-time FCD visualization with vehicle id

City travel planner

The travel planner suggests the best routes from traveling from location A to B based on the current FCD received from the vehicles traveling the road network. The following information is required from the user for the travel planning:

- Start location
- Destination
- Date and time of travel and if it is the start time or time of arrival

Planning can occur based on historical data, real-time data or a combination of both. The MapMatcher component did not support travel planning at the time this work was done. However the implementation in this work is done based on a sample data in the same format as would be received from the MapMatcher. When the MapMatcher starts supporting this functionality, connection the Visualizer to the real source of data would be trivial.

Figure 31 displays the travel planner in full-screen mode with the sample data.

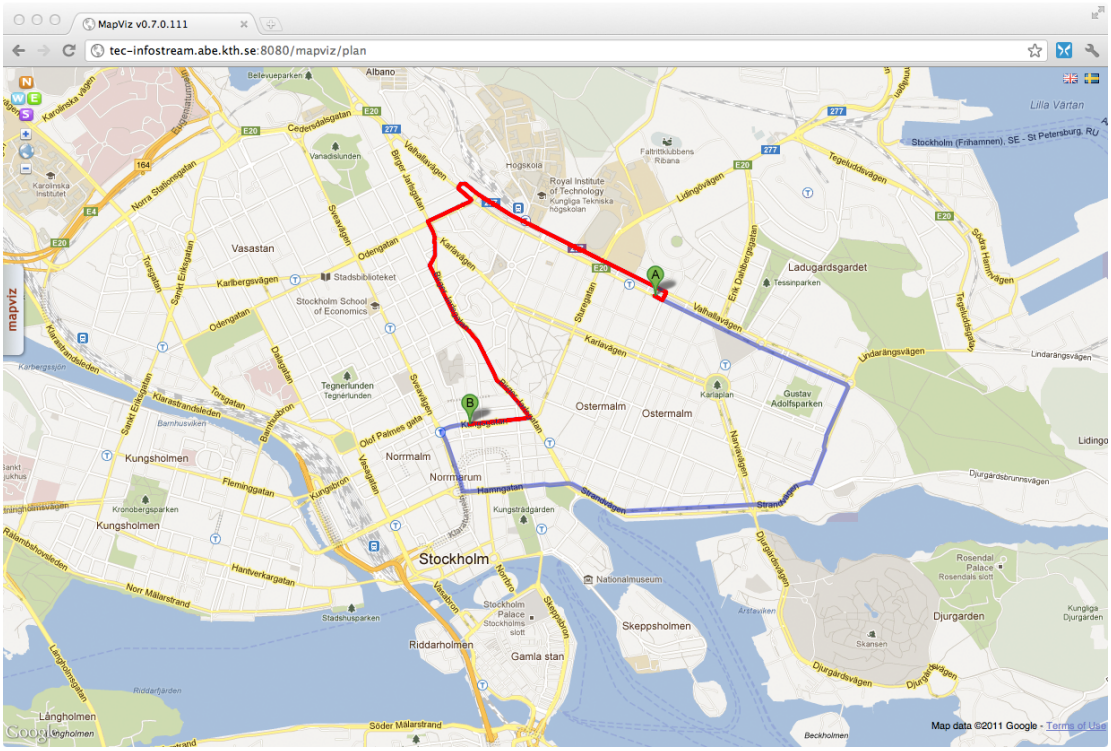


Figure 31 - Travel planner

4.6. Performance

Performance is a key factor in MapViz architecture since it directly affects the achievable animation smoothness. Animating the taxi icons is the result of frequently refreshing their locations on the map. Therefore increasing the refresh frequency leads to a more fluid animation and improves the user experience.

The refresh frequency is limited by how fast the system can provide result for an arbitrary request (response time), which in turn is affected among other factors by the systems architecture and network latency. If the refresh requests sent to the system with a faster rate than the system can generate responses, the performance of the system starts to degrade which translate to frequent pauses in the animation.

The response time of the presented architecture on an average development machine[30] is around 150 ms. An appropriate refresh rate for such a response time would be every 200 ms.

The provided numbers are result of single-user testing. When multiple concurrent users use the system, the number of requests is roughly increased by the same factor and should be considered for determining the refresh rate.

[30] CPU Intel Dual Core 2.2 GHz and RAM 2 GB

The current design can be scaled up at both Data Services and Visualizer level to reach the desired response rate for the required number of concurrent requests. Both components are intentionally designed stateless which means requests contain all the information required for processing and providing a response. In other words, the server does not need to know about the previous requests sent by the client. Given that more than one server hosting a component, this in essence means a client can send any of the consequent requests to any of the hosting servers and receive correct responses. Taking advantage of this principle, both Visualizer and Data Services components scale almost linearly by adding more servers hosting those components and a load balancer in front of them.

Figure 32 demonstrates the proposed server topology with four servers hosting MapViz Visualizer and four servers hosting MapViz Data Services in order to accommodate higher number of concurrent users.

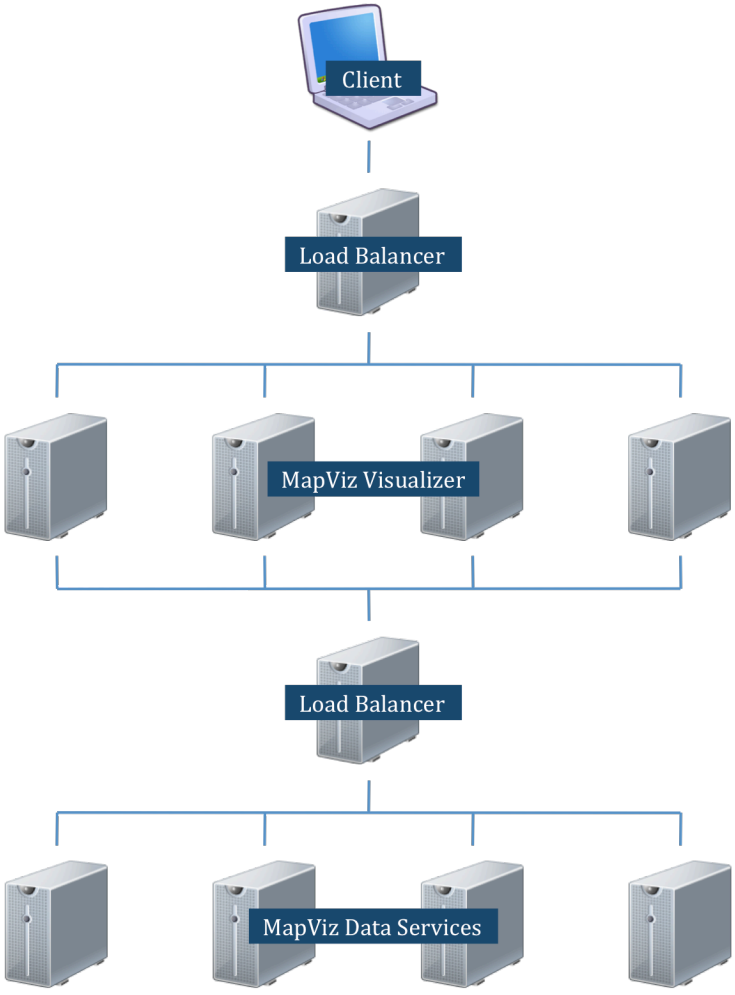


Figure 32 - MapViz server topology for higher number of concurrent users

5. Conclusion

The solution provided in this thesis is MapViz visualization platform, which is capable of visualizing streams of data as an overlay on a map. The platform is divided into two applications Data Services and Visualizer. MapViz Data Service is responsible for retrieving data from variety of sources and then transforms and enriches it so that it becomes presentable. The Visualizer is responsible for consuming the data provided by Data Services and renders it on map. The separation of responsibility is done in a way that makes it possible to feed other sources of data to the Visualizer as long as it conforms to the defined contact. Furthermore, several Visualizers targeting different requirements can be added to the platform to produce specific visualizations of the same data.

MapViz platform implemented as a result of this thesis supports a variety of visualization scenarios among others the objectives mentioned in chapter 1. Nevertheless, the current version of the platform has focused merely on the required scenarios for the purpose of this thesis and further development is left to the future work.

6. Future Work

This thesis is a starting point for a framework capable of digesting and visualizing geographic data without binding itself to any particular provider or vendor. There are several extension points to this work that can be investigate further to enrich the functionality provided by MapViz.

Despite the possibility of switching between map providers, in the current implementation the switch needs changes in the configuration of Visualizer at deploy time. This can be further delegated to the user through a control panel so that he can change providers on-the-fly as he sees fit. Furthermore, more geocoding plug-ins should be implemented and added to the geocoding layer to decrease the chance of running out of geocoding request quota.

Smoothness of animations in MapViz is directly dependent on how quickly the system can provide a response. Therefore, performance optimization on each step of request processing pipeline is of great value to achieve a more pleasant user-experience. The best place to start is possible optimizing KML generation by benchmarking alternative XML processors such as JIBX. Investigation of possible caching solutions for Visualizer is another area that can significantly affect the performance.

The current implementation of MapViz doesn't provide any type of security for limiting access to the service. Authentication and authorization needs to be implemented to be able to restrict usage of Data Service and Visualizer particularly if it is accessible from the Internet.

As a part of this work, only on Visualizer implemented to demonstrate the concepts and enable visualization in a desktop web browser. Other visualizers can be implemented and added to MapViz for supporting mobile web, iPhone, iPad, Android and other platforms.

As discussed in previous chapters, real-time in his work is not to the meaning of the word and visualization happens with minor delays. Further investigation is required for the map-matching component to be able to remove the delay as much as possible and provide a real-time state of the network. On the other hand it can be valuable to enable the Visualizer to visualize FCD based on historic data in other words, replaying a certain point in the past.

The Visualizer application can be made richer by adding road hazards information as they are received in addition to color-coding the vehicle trails based on average speed of the vehicle and max speed of the links. Taking into account factors like maximum allowed speed on a link as well as current vehicle speed can optimize calculations of vehicle positions on the links and provide a more realistic animation of the vehicles on the map.

7. Bibliography

Ahmadi, P. (2010, November). Floating Car Data and Applications. Stockholm, Sweden: KTH, School of Architecture and the Built Environment.

Butler, H., Daly, M., Doyle, A., Gill, S., Schaub, T., & Schmidt, C. (2008, June 28). *The GeoJSON Format Specification*. Retrieved June 2, 2011, from GeoJSON: http://geojson.org/geojson-spec.html

Caldwell, H. (2002, June). Border Wizard: Enhancing Border Security and Productivity. *Freight & International Trade Conference* . Pittsburgh, PA: TRB Ports.

Coupling. (n.d.). Retrieved June 5, 2011, from Wikipedia: http://en.wikipedia.org/wiki/Coupling_(computer_science)

Fastenrath, D. U. *Floating Car Data on a Larger Scale.* Düsseldorf: DDG Gesellschaft für Verkehrsdaten mbH.

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures.

Friedman, V. (2008, January 14). *Data Visualization and Infographics.* Retrieved June 3, 2011, from Smashing Magazine: http://www.smashingmagazine.com/2008/01/14/monday-inspiration-data-visualization-and-infographics/

Friendly, M. (2009). Milestones in the history of thematic cartography, statisticalgraphics, and data visualization.

GIS Cookbook. (2007, 10 25). Retrieved 01 13, 2012, from Housing and Land Use Regulatory Board: http://www.cookbook.hlurb.gov.ph/book/export/html/34

GML. (2007). *OpenGIS® Geography Markup Language (GML) Encoding.* Open Geospatial Consortium Inc.

Goel, S., Imielinski, T., Ozbay, K., & Nath, B. (2003). Grassroots: A Scalable and Robust Information Architecture. *Dept of Computer Science Rutgers University Tech Rep DCSTR523* , 1-22.

(2006). In C. Hixon, *NCHRP Synthesis 361: Visualization for Project Development, A Synthesis of Highway Practic.* Washington D.C.: Transportation Research Board.

Howard, J., Weisberg, P., & Pack, M. (2008). Four-Dimensional, Real-Time Automatic Vehicle Location Visualization System. *Transportation Research Board 87th Annual Meeting.* Washington D.C.

Kerner, B., Demir, C., Herrtwich, R., Klenov, S., Rehborn, H., Aleksic, M., et al. (2005, September 13-15). Traffic State Detection with Floating Car Data in Road Networks. *Intelligent Transportation Systems* , pp. 44-49.

Kimpel, T. J. (2007, Feb). Data Visualization as a Tool for Improved Decision Making within Transit Agencies. Seattle, WA: Transportation Northwest (TransNow), University of Washington.

KML. (n.d.). Retrieved June 02, 2011, from Wikipedia: http://en.wikipedia.org/wiki/Keyhole_Markup_Language

Koutsopoulos, H., Rahmani, M., Güc, B., Biem, A., Bouillet, E., Feng, H., et al. (2010). Real-Time Traffic Information Management. *IEEE Data Eng. Bull. , 33*, 64-68.

Lawson, C., & Jonnalagadda, S. (2006, July). SWATH Analysis: Using Firm-level Data to Understand truck Trips on New York State Trade Corridors. Prepared for the Office of Policy and Strategy, New York State Department of Transportation.

Manore, M. (2008). Visualization Education and Training. *ACM SIGGRAPH 2008 classes (SIGGRAPH '08)* .

OpenLayers. (2011, December 18). Retrieved December 18, 2011, from OpenLayers: http://openlayers.org/

Pack, M., Weisberg, M., & Bista, S. (2007). Wide-Area, Four-Dimensional, Real-Time Interactive Transportation System Visualization. *Journal of the Transportation Research Board* (2018), 97-108.

Rahmani, M., Koutsopoulos, H. N., & Ranganathan, A. (2010). Requirements and potential of GPS-based floating car data for traffic management: Stockholm case study. *Intelligent Transportation Systems (ITSC)* , 730-735.

Rass, S., Fuchs, S., Schaffer, M., & Kyamakya, K. (2008). How to protect privacy in floating car data systems. *Proceedings of the fifth ACM international workshop on VehiculAr Inter-NETworking* (pp. 17-22). San Francisco: ACM.

Rhyne, T.-M. (2007). Visualization and the larger world of computer graphics. *TR News* (252), 20-23.

Rose, S. (2005). Aviation Visualization and Analysis.

Schmitt, R., Dronzek, R., & Rohter, L. (2007). Visualization to Solve Problems in Freight Transportation. *TR News* (252), 15-19.

Schwabe, C. (2007). *The Forrester Wave: Software Change And Configuration Management, Q2 2007.* Forrester.

Shankland, S. (2008, April 14). *Google Mapping Spec Now an Industry Standard*. Retrieved June 2, 2011, from CNet News: http://news.cnet.com/8301-10784_3-9917421-7.html

Stevens, W. P., Myers, G. J., & Constantine, L. (1974). Structured design. *IBM Systems Journal , 13* (2), 115-139.

Turksma, S. (2000). The Various Uses of Foating Car Data. *Road Transport Information and Control* , 51-55.

Williams, E. (2011, 4 25). *Aviation Formulary v1.46*. Retrieved 1 14, 2012, from Ed Williams' Aviation: http://williams.best.vwh.net/avform.htm

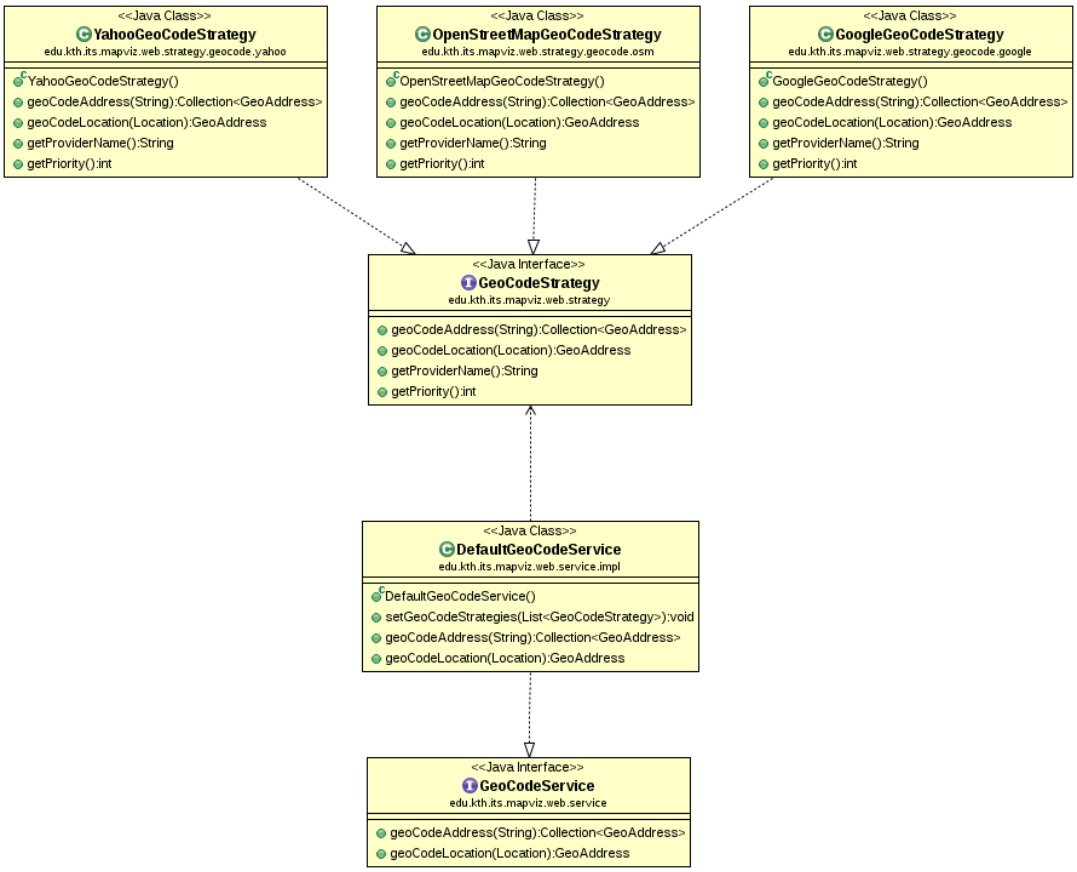
Appendix A - MapViz Data Services WADL

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
 <doc xmlns:jersey="http://jersey.dev.java.net/" jersey:generatedBy="Jersey: 1.4
09/11/2010 10:30 PM"/>
  <resources base="http://tec-infostream.abe.kth.se:8080/mapvizservice/">
    <resource path="/vehicle">
      <resource path="/ids">
        <method name="GET" id="getCarIds">
          <response>
            <representation mediaType="application/json"/>
          </response>
        </method>
      </resource>
      <resource path="/trails">
        <method name="GET" id="getCarTrails">
          <request>
<param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="query"
name="ids"/>
<param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="query"
name="bound"/>
<param xmlns:xs="http://www.w3.org/2001/XMLSchema" default="false" type="xs:boole
an" style="query"name="accumulative"/>
          </request>
          <response>
            <representation mediaType="application/json"/>
          </response>
        </method>
      </resource>
      <resource path="/positions">
        <method name="GET" id="getCarPositions">
```

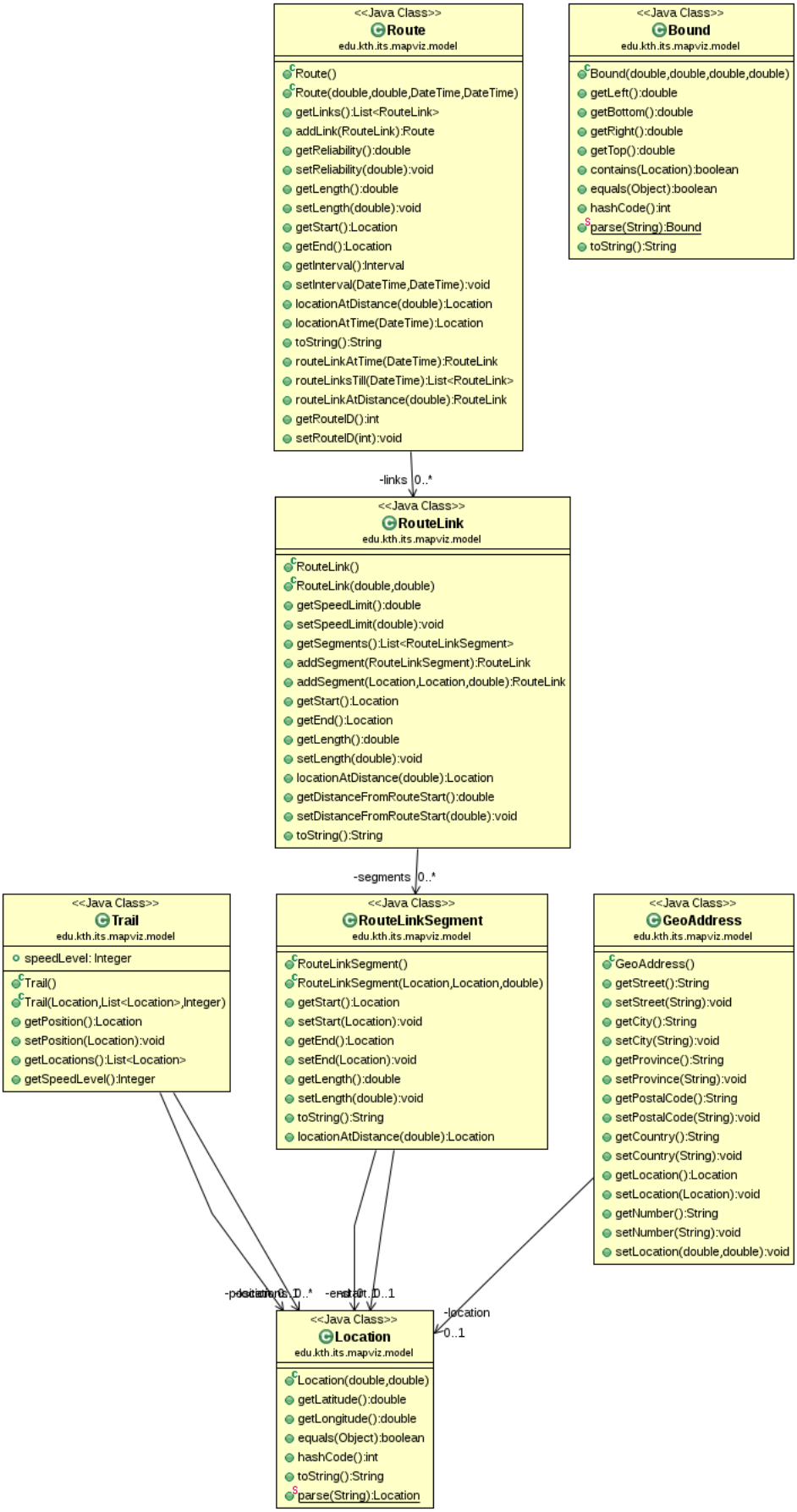
```
            <request>
 <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="query"
name="ids"/>
 <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="query"
name="bound"/>
            </request>
            <response>
              <representation mediaType="application/json"/>
            </response>
          </method>
        </resource>
      </resource>
    </resources>
</application>
```

Appendix B – Class Diagrams

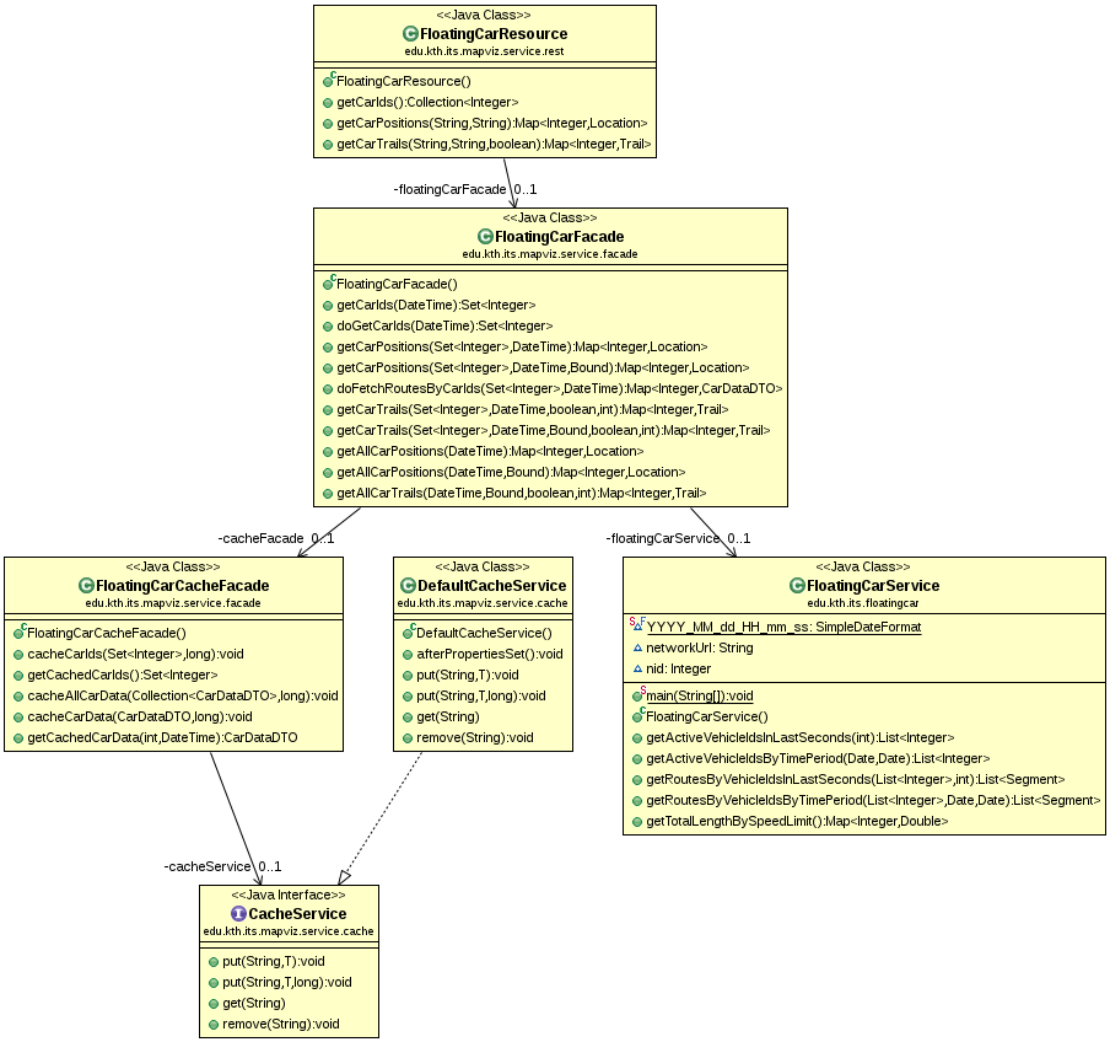
GeoCoding Layer Class Diagram



Data model class diagram



Data Service class diagram



Visualizer class diagram