GENERIC ANIMATION TOOL FOR TRAFFIC SIMULATION

A Thesis Presented to the Department of

Computer Science

African University of Science and Technology

In Partial Fulfilment of the Requirements for the Degree of

Master of Science

Submitted by

Olayiwola, Joy Ugonma

Abuja – Nigeria

June 2016

GENERIC ANIMATION TOOL FOR TRAFFIC SIMULATION

By

Olayiwola Joy Ugonma

A THESIS APPROVED BY THE COMPUTER SCIENCE DEPARTMENT

RECOMMENDED: _____

Supervisor, Professor Mamadou Kaba Traore

_____

Head, Department of Computer Science

APPROVED: _____

Chief Academic Officer

_____

Date

ABSTRACT

Traffic simulation has become one of the most used approaches for traffic analysis in support of the design and evaluation of traffic systems. Although traffic flow models have been applied for almost a century to describe, simulate and predict traffic, digital computer programs to simulate traffic flow have been developed from the 1950s. With the increasing power of computers, simulations began to incorporate animation techniques. These animation techniques and visualizations allowed viewing the overall performance of a traffic system design while providing an excellent means of communicating the result patterns from a simulation model to officials, decision makers and the general public in a meaningful way.

This work presents the design of an animation/visualization tool for road traffic simulation, which is independent (stand-alone/separate from a traffic simulator) and generic (that is, can visualize output data from any traffic simulator). This tool implements Google Maps as its background, thereby enabling users to view the animation of a simulation output on any target road. The source data for the animation is an XML file which holds vehicle information.

ACKNOWLEDGEMENT

To God be the glory for the great things He has done.

I will like to express my gratitude to **Professor Mamadou Kaba Traore** for his support, guidance and encouragement throughout this work. Working with him has been a challenging yet rewarding experience.

My appreciation also goes to Mr. Osondu Onwuzurigbo (JOC) for his support, advice and encouragement, especially when I was at my wits' end.

Also, my appreciation goes to the African University of Science and Technology (AUST), Abuja community for providing support and a wonderful research environment.

My appreciation also goes to my Director-General, NASRDA, Prof. Seidu Mohammed, for granting me leave of study from the office. I do not take it for granted.

To my husband and children (Olukemi and Olufemi), you have been most supportive. Thank you so much for your patience and understanding.

To my family – my moms (especially for their consistent prayers), sisters, brothers, cousins, and everyone who helped in the home, thank you so much for your support and understanding. You all made it easy for me, knowing you got my back.

To my colleagues and classmates at AUST, it has been an awesome journey working together. I look to forward to continued bonding.

To my Director, line Managers and colleagues at NASRDA, thank you all for your encouragement.

To my leaders in FWC Lugbe – my Pastor, Deacons and Coordinators – thank you so much for your prayers, calls, care and encouragement.

To my Worship Team family of FWC Lugbe, thank you for your support, prayers and encouragement.

May God bless and reward you all.

DEDICATION

I dedicate this thesis to the Almighty God, my everything. I would not have made it without Him.

To my brother, Kingsley C. Osuocha; you looked forward to this day, but death dealt me a great blow. I miss you.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
RESEARCH CONTEXT

1.1 INTRODUCTION

Simulations of any system give users and decision makers an opportunity to appraise alternative strategies of the system before implementing them in the field. Digital computer programs to simulate traffic flow have been developed from the 1950s. The increasing power of computer technologies, the advances in software engineering and the advent of intelligent transport systems prompted traffic simulation to be one of the most used approaches for traffic analysis in support of the design and evaluation of traffic systems. The ability of traffic simulation to emulate the time variability of traffic phenomena makes it a unique tool for capturing the intricacy of traffic systems (Barcelo, 2010).

Numerous research activities that have been carried out on traffic systems have concentrated on modelling, simulation and visualization/animation of rural and urban traffic by taking advantage of advances in computer technology, either to assess alternatives in traffic management or to assist traffic system construction in urban development. The physical dissemination of traffic flows can be specifically depicted using traffic flow models. By utilizing different traffic simulation models, one can simulate large scale real-world situations in great detail. Depending on the level of detailing, traffic flow models are classified into macroscopic, mesoscopic and microscopic models. Brief descriptions of these model types are illustrated below.

1.1.1 MACROSCOPIC MODELS

Macroscopic models view the traffic flow in general. That is, these models are usually based on the continuous traffic flow theory whose objective lies in observing/assessing the time-space development of the variables describing the traffic flows. These variables are volume, speed and density, which are assumed to be defined at every instance in time t and every point in space x. Macroscopic models gained interest since the 1960s. Examples of these types of models include FREFLO – FREeway FLOw (Payne, 1979), METANET – Modèle d'Ecoulement du Trafic Autoroutier: Network (Messmer et al., 1990-9).

1.1.2 MICROSCOPIC MODELS

This type of model gives attention to individual vehicles and their interactions. It models driver/vehicular actions like acceleration/deceleration, lane changes in response to the surrounding traffic. Such models seek to answer questions like the nature of a driver's response to an event, measuring a drivers' sensitivity, etc. Microscopic models cover car-following models for single-lane traffic, and incorporate lane-changing models for multi-lane traffic. Examples are FRESIM – FREeway micro-SIMulator, CORSIM – CORridor SIMulation, VISSIM – a German acronym for "Traffic in Towns – Simulation", TRANSIMS – TRansportation Analysis and SIMulation System (Sharon et al., 2001).

1.1.3 MESOSCOPIC MODELS

This type of model comprises, in some ways, microscopic and macroscopic aspects of traffic flow models. As a result, they are computationally more efficient than microscopic models. But observing the trend, mesoscopic models have not been receiving much research attention lately. Examples are DYNAMEQ – Dynamic EQuilibrum, DYNMIT (Barcelo, 2010), etc.

1.2 VISUALIZATION FOR ROAD TRAFFIC SIMULATORS

The visualization aspect of the research activities in road traffic simulation are receiving substantial interest from both academia and industry due to huge amounts of data stored in traffic and transportation databases or the amount of data generated by simulation models (Shekar et al., 1997). Data is generated from simulation models, especially when it is large, is difficult to easily interpret for planners, policy makers, even the modellers running the simulation. This gave birth to the integration of visualization and its techniques to traffic modelling and simulation, thereby enabling the extraction of useful information from the large mass of data.

Most of the existing visualization tools are either integrated with the traffic simulators or designed for specific traffic simulators. For example, METROPOLIS is a visualization tool designed for TRANSIMS simulation output (https://sourceforge.net/projects/transims-metro) while SUMO (Simulation of Urban MObility – an open-source microscopic, multi-modal traffic simulation) is that it has its visualizer/GUI embedded in its software. A number of road design techniques have been implemented in road traffic visualization tools ranging from graphs and charts to 2D/3D road network designs and maps (Open Street Map, Google

Earth). Very few visualizers are stand-alone (that is, not a traffic simulator and independent of the simulation model or software used) software that can accept data describing the road network and vehicle trajectories and still render its output in real time. One of such is vtSim.VIEW, a module developed for the vtSim (Validating environment for Traffic Simulation) data and simulation management framework (Wenger et al., 2013).

Based on the search and reviews carried out in the course of this work, it was observed that most road networks of these visualizers were manually designed or constructed with various mathematical calculations performed in order to mimic an almost accurate geometry of the road network being observed. For the few visualization tools that use or incorporate Google Maps, Google Earth or OpenStreetMap as background, the map data is either converted to a simulator-specific road network format using a tool in its package (SUMO) or the map is used as a background photo and the model is overlaid on the photo (VISSIM) (Matthew et al., June 2007).

1.3 OBJECTIVE

This project aims to design an animation/visualization tool for road traffic simulation, that is independent, generic (that is, can visualize output data from any traffic simulator), and has a realistic feel of traffic observation, as we will incorporate Google Maps in its dynamic form; the user only needs to pan to the road being observed on the map. The data source for this visualization will be XML based, as XML files can be efficiently read, exchanged and their format defined in a standardized way using XML schema.

1.4 STRUCTURE OF WORK

We organize our work as follows: Chapter 2 is the literature review on the Visualizers of Traffic Simulation and their implementation techniques, while Chapter 3 presents our design methodology. Chapter 4 addresses the implementation and testing of our application. In Chapter 5 we conclude our work and discuss the limitations encountered and give the direction of future work.

CHAPTER 2
LITERATURE REVIEW

2.1 INTRODUCTION TO TRAFFIC M&S

Eighty years ago, Bruce Greenshields presented the first traffic flow model at the Annual Meeting of the Highway Research Board. Since then, many models and simulation tools have been developed. Traffic flow models have been applied for almost a century to describe, simulate and predict traffic. The first model, by Greenshields, showed a relation between the distance between vehicles and their speed. Later, dynamics were included in the models and models were applied for predictions. Now, traffic flow simulation tools were used for long-term planning as well as for short-term predictions based on actual traffic data, dynamic traffic management, evacuation planning, to mention a few of its uses and benefits (TRC, 2015).

Figure 2.1 shows the historical development of traffic flow models as a model tree. From this diagram, it is observed that all traffic flow models have one common root: the fundamental relation (or fundamental diagram – FD). The other three families consist of the microscopic, mesoscopic and macroscopic models. After the introduction of the fundamental relation in the 1930s, microscopic and macroscopic models were introduced simultaneously in the 1950s. Mesoscopic models are about a decade younger. Microscopic models distinguish and trace the behaviour of each individual vehicle while macroscopic models aggregate vehicles and traffic and they are usually described as a continuum. Mesoscopic models are categorized in between microscopic and macroscopic models as their (mesoscopic) aggregation level is between both (micro- and macroscopic models).

Traffic modelling theories seek to describe in a precise mathematical way the interactions between vehicles and their operators (the mobile components) and the infrastructure (the immobile components). The infrastructure consists of the road network and all its operational elements like control devices, signs and markings.

In the figure below, grey lines indicate descent; black dots indicate publications; black lines indicate that the model has (or multiple very similar models have) been published multiple times (Kessels, 2013). (Most labels are omitted for readability).

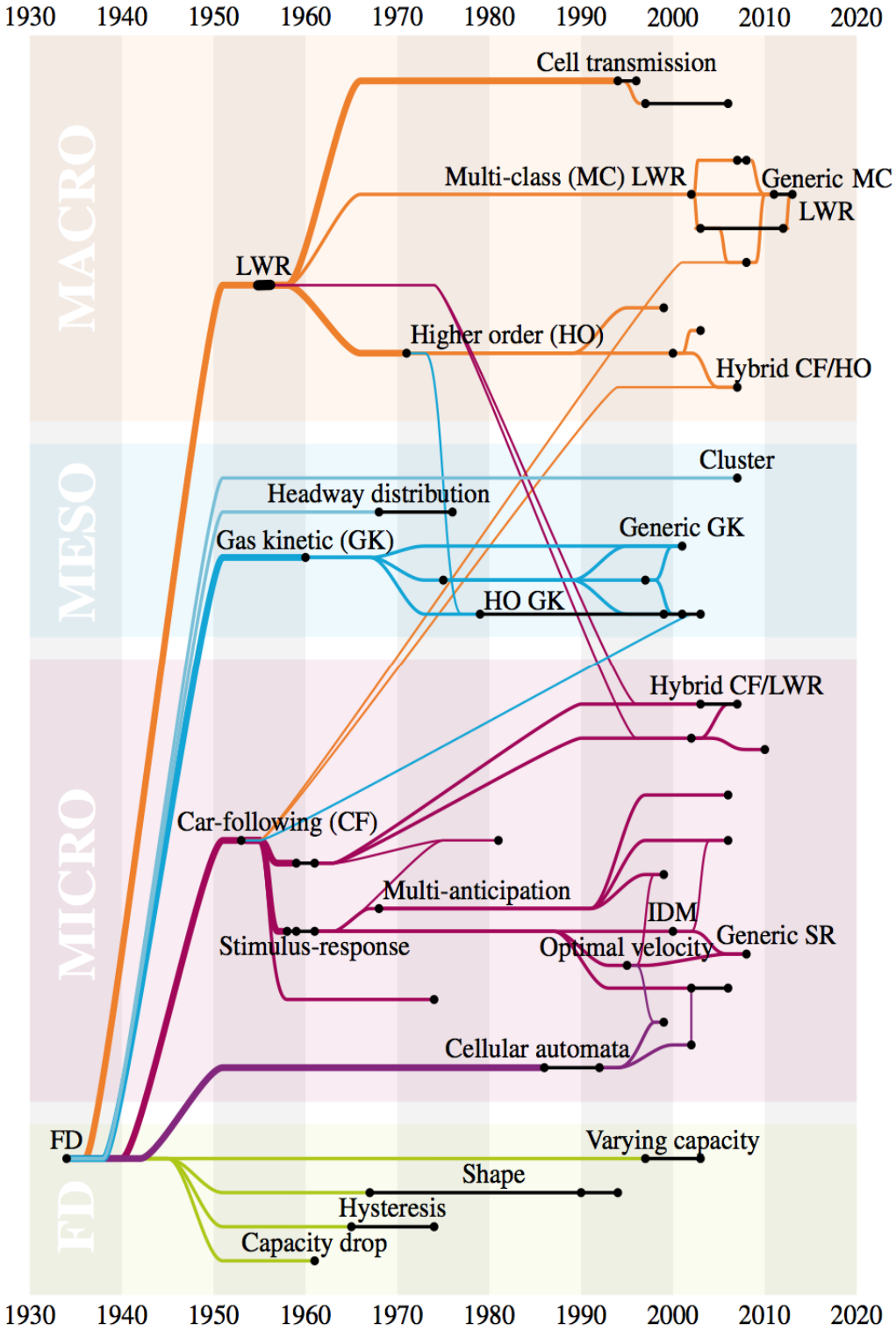


Figure 2.1: Genealogical Tree of Traffic Flow Models

Digital computer programs to simulate traffic flow have been developed from 1950 onwards. Early simulation model developers of this period had to deal with an adverse computing environment. Not only were computers in limited supply and computer time very costly, but software developers also had to deal with severe computer storage and programming constraints. While the limited availability of computing equipment restricted the development of simulation software in the 1950s, theoretical developments were taking place which would profoundly promote the development and use of traffic simulation in the future. Pioneers involved in the emergence of transportation engineering in this decade include:

- John Glen Wardrop, an English Mathematician and transport analyst, who in 1952 articulated principles that have been widely used to simplify the mathematics associated with routing in traffic models (www.wisdot.info). His equilibrium laws form the basis for traffic assignment and the present application of simulation-based network modelling (TRC, 2014).
- Sir Michael James Lighthill (a British applied mathematician) and Gerald B. Whitham (an American applied mathematician) developed together the fluid flow analogies of traffic flow in 1955 – comparing traffic flow on long crowded roads with the flood movements in long rivers. A year later, P.I. Richards complemented the idea with the introduction of "shock-waves on the highway" in 1956, thereby completing the so-called LWR model. This LWR theory forms the basis for most macroscopic simulation models (Wikipedia, TRC 2014).
- R.E. Chandler, Robert Herman and E.W. Montroll put forward, in 1958, the first prototype of a car-following model which led to the GHR model/formula by D.C. Gazis, Robert Herman and R.W. Rothery in the late 1950s and early 1960s. The car-following model forms the core of microscopic simulation models (Brackstone et al., 1998).

Traffic simulations were first developed, independently, by different countries, and often to investigate a small problem domain. The first simulations were developed on large government and university mainframe computers. Recent simulations are a combination of the work of many earlier simulations from many countries and therefore often cover a much wider problem domain (Fotherby, 2002). The increasing power of computers and computing technology enabled simulations to begin to incorporate animation techniques. This innovation allowed viewing the overall performance of a traffic system design while providing an

excellent means of communicating the result patterns to officials, decision makers and the general public.

2.2 GRAPHICAL TRAFFIC SIMULATIONS

Simulation and visualization are naturally connected. A simulation advances the state of a modelled system through time while visualization provides an abstract visual rendering of the state of that system at any point in time. Visualization offers one of the most promising means to convey information from a simulation model to decision makers in a meaningful way. In recent times, as outlined by Charles Macal, simulation and visualization can be viewed as encompassing four broad areas of research activity (Charles, 2001).

- Animation (2D/3D) generated from the simulation: Animations are run as post-processors to simulations to depict the simulation results. The animations can be created with varying degrees of sophistication and degrees of realism. Important design decisions regarding an animation include the type and detail of information transferred from the simulation, and the benefits obtained relative to the cost of constructing the animation. The user is not able to interact with the simulation through the animation.
- Visual Interactive Simulation (VIS): In this case, the user interacts with all phases of the simulation through a visual interface. VIS is a simulation method that lets decision makers see what the model is doing and how it interacts with the decisions made, as they are made. VIS uses animated computer graphics display to present the impact of different decisions. It differs from regular graphics in that the user can adjust the decision-making process and see the results of the intervention.
- Graphical interfaces for building simulation models: Visual languages have been developed along with systems for creating visual primitives for selected domains. These visual tools are expressive enough to be used to assemble complete simulation models and to specify alternative simulation runs. They use a high-level graphical representation formalism based on the activity-cycle diagrams to define simulation models in an interactive mode.
- Immersive simulation and virtual simulation environment: Virtual reality refers to a set of techniques in which one interacts with a synthetic (“virtual”) environment that exists solely in the computer. In the typical conception of virtual reality, the

representation of the synthetic environment is fed fairly directly to the eyes, ears and possibly hands. A variant of virtual reality is often called “Visualization”, which involves presentation of 3D structures (such as road structures and buildings) in ways that maximize learning and understanding of the simulation.

One of the best ways for non-technical users to understand the results of a traffic simulation is to actually view the simulation traces. To achieve this, graphical representation of the traffic simulation is a good method to examine what exactly happened on the road and at what periods in time it happened. Just as the advent of traffic simulation has been researched and simulators developed over the years, with improvements being implemented for each new version or package, so also the visualization part of it has witnessed tremendous growth. Graphical presentation of simulation appeared in the late 1960s (Matti, 1999) (see Figure 2.2).

In recent times, most simulation packages now include different graphical front ends; some packages exist which include a non-graphical simulation that only produces output files and statistics by which the events within the traffic network can be measured or determined.

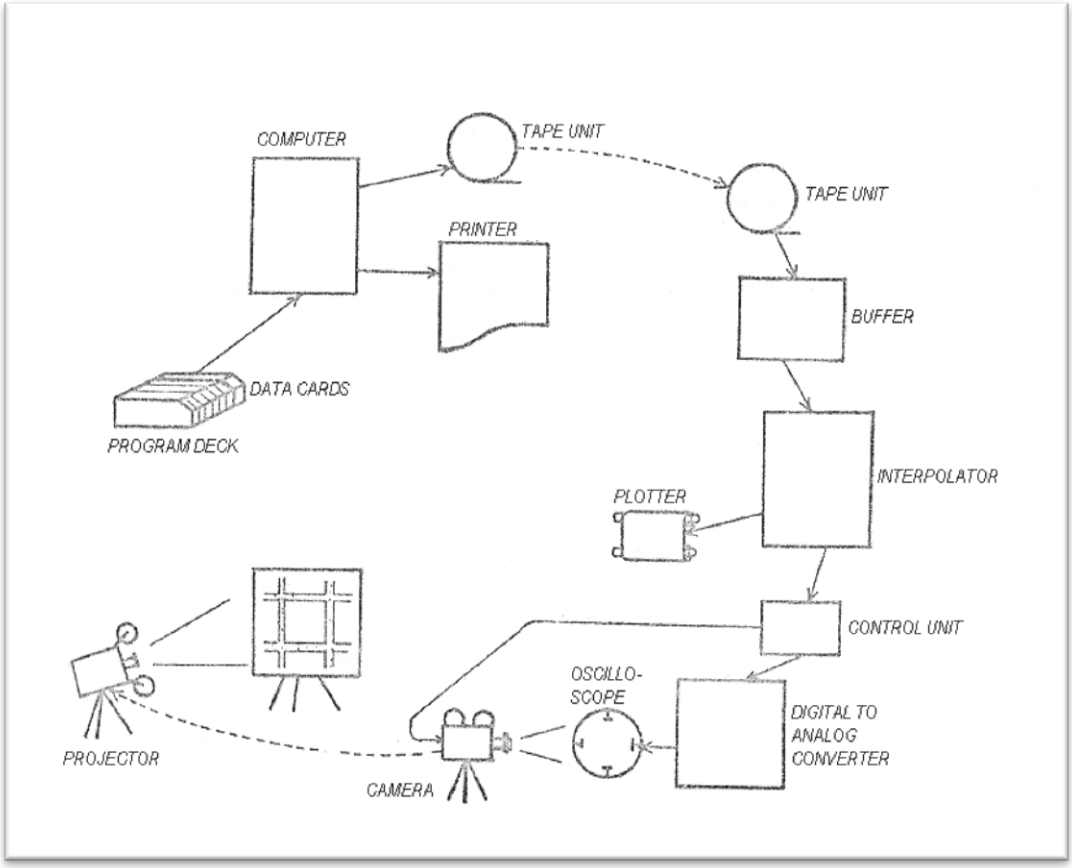


Figure 2.2: A Graphical Representation of Simulation Result in the Late 1960s

While comparing some modern traffic simulation software packages that implemented the macroscopic and microscopic modelling strategies, Kotusevski noted that among the simulation applications that include a graphical front end which visually represents the traffic simulation carried out, there still exist major differences between the qualities of these graphical representations (Kotusevski et al., 2009). The key distinction of the graphics is that some of the packages had only two-dimensional representation while others had the option of three-dimensional representation. For the package with two-dimensional graphic representation, we have an example like Treiber's Microsimulation of Road Traffic, a personal software project created by Martin Treiber and used in his research of traffic dynamics and traffic modelling (see Figure 2.3). Figure 2.4 presents an example of a 3-dimensional traffic visualizer called Quadstone Paramics Modeller. It is a modular suite of microscopic simulation tools providing a powerful, integrated platform for modelling a complete range of real-world traffic and transportation problems. It uses the available graphics libraries to preview buildings around the traffic network, as well as traffic members such as cars, trucks, public transport and pedestrians.

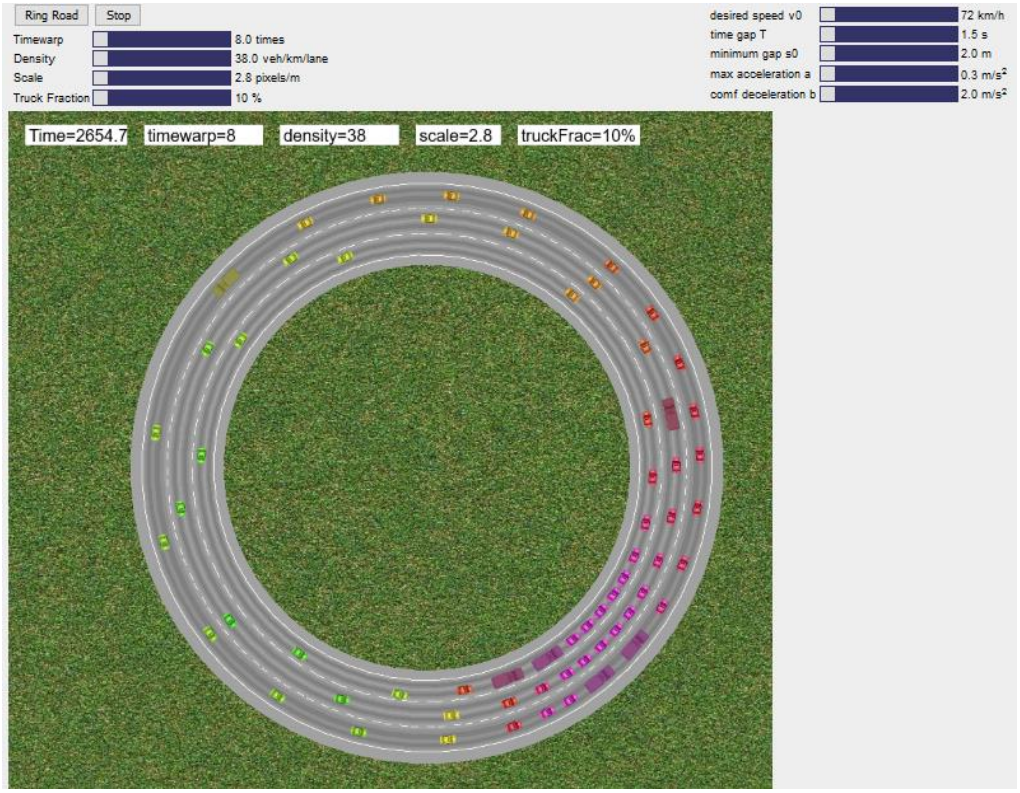


Figure 2.3: Treiber's Microsimulation of Road Traffic Applet

Figure 2.4: Paramics Modeller Traffic Simulation

Since our work will focus on a road traffic visualizer, let us examine a commercial traffic simulator that incorporated animation in its package and two related works.

2.2.1 SUMO

"**S**imulation of **U**rban **MO**bility", or "SUMO" for short, is an open-source, purely microscopic, space-continuous and time-discrete traffic flow simulation. It is written in C++. The implementation of SUMO started in 2001, with a first open-source release in 2002. In the past 10 years, SUMO has evolved into a fully featured suite of traffic modelling utilities including a road network importer capable of reading different source formats, demand generation and routing utilities, which use a high variety of input sources (origin destination matrices, traffic counts, etc.), a high-performance simulation usable for single junctions as well as whole cities including a "remote control" interface to adapt the simulation online and a large number of additional tools and scripts.

SUMO is not only a traffic simulator, but rather a suite of applications, which help to prepare and to perform the simulation of a traffic scenario. The SUMO application utilizes own configurations for road networks and traffic demand where both inputs have to be imported or generated from existing sources of various kinds (Krajzewicz et al., 2012). All SUMO applications support XML validation for their inputs. The next section briefly summarizes features of SUMO.

2.2.1.1 ROAD NETWORK GENERATION

SUMO road networks are encoded as XML files. A SUMO network file describes the traffic-related part of a map, the roads and intersections the simulated vehicles run along or across. Nodes, usually named "junctions" in the SUMO context, represent intersections, and "edges" represent roads or streets. Specifically, the SUMO network contains the following basic information:

- Every street (edge) as a collection of lanes, including the position, shape and speed limit of every lane
- Traffic light logics referenced by junctions
- Junctions, including their right-of-way regulation
- Connections between lanes at junctions (nodes).

Intersections consist of a position, a shape and right-of-way rules, which may be overridden by a traffic light. Edges are unidirectional connections between two nodes and contain a fixed

number of lanes. A lane contains geometry, the information about vehicle classes allowed on it, and the maximum allowed speed. Therefore, changes in the number of lanes along a road are represented using multiple edges. Other than this basic perspective on a road network, SUMO road networks incorporate traffic light plans, and connections between lanes across intersections describing which lanes can be used to reach a subsequent lane.

SUMO road networks can be generated either by using an application in the suite named NETGENERATE or by importing a digital road map using NETCONVERT. To create a SUMO network file, NETCONVERT helps to generate it from maps in other formats and NETGENERATE constructs a new map with simple geometries. SUMO performs its simulation directly in the map of this file. The SUMO network file, though in XML, which makes it easy to read, is not meant for direct editing. It must be converted to the SUMO native XML descriptions with NETCONVERT instead, and then a user can process these files by hand). NETGENERATE builds three different kinds of abstract road networks: Manhattan-like grid networks, circular "spider-net" networks, and random networks (See Figure 2.5).

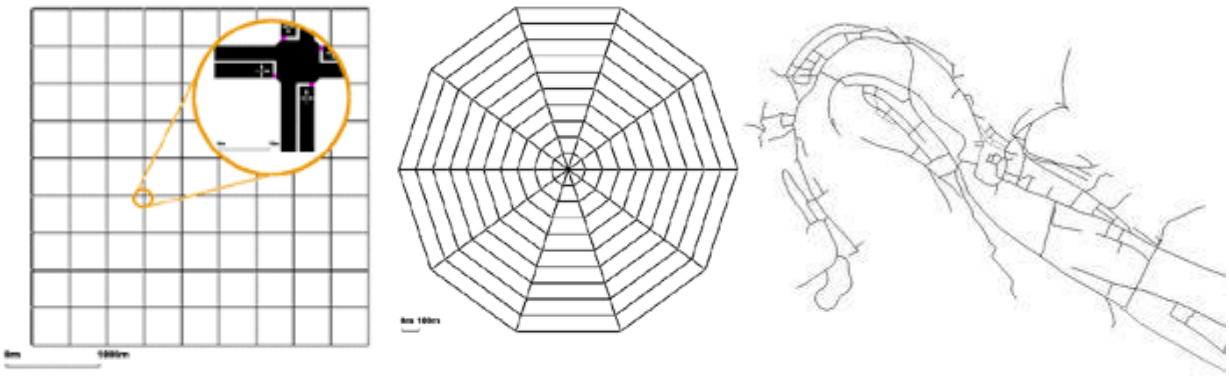


Figure 2.5: Road Networks Built Using NETGENERATE

From left to right: Grid (Manhattan), Spider and Random network

NETCONVERT is able to read road networks from other non-SUMO sources like VISUM, TIGER, ArcView shape files, VISSIM, Robocup Rescue League folders, OpenStreetMap, and a "native" XML representation of the road network graph (Barcelo, 2010). Below is a definition of a road network, by their nodes and edges in XML.

```
<nodes>
<node id="0" x="0.0" y="0.0" type="traffic_light"/>

<node id="1" x="-500.0" y="0.0" type="priority"/>

<node id="2" x="+500.0" y="0.0" type="priority"/>

<node id="3" x="0.0" y="-500.0" type="priority"/>

<node id="4" x="0.0" y="+500.0" type="priority"/>

<node id="m1" x="-250.0" y="0.0" type="priority"/>

<node id="m2" x="+250.0" y="0.0" type="priority"/>

<node id="m3" x="0.0" y="-250.0" type="priority"/>

<node id="m4" x="0.0" y="+250.0" type="priority"/>
</nodes>

<edges>

<edge id="1fi" from="1" to="m1" priority="2" numLanes="2" speed="11.11"/>

<edge id="1si" from="m1" to="0" priority="3" numLanes="3" speed="13.89"/>

<edge id="1o" from="0" to="1" priority="1" numLanes="1" speed="11.11"/>

<edge id="2fi" from="2" to="m2" priority="2" numLanes="2" speed="11.11"/>

<edge id="2si" from="m2" to="0" priority="3" numLanes="3" speed="13.89"/>

<edge id="2o" from="0" to="2" priority="1" numLanes="1" speed="11.11"/>

<edge id="3fi" from="3" to="m3" priority="2" numLanes="2" speed="11.11"/>

<edge id="3si" from="m3" to="0" priority="3" numLanes="3" speed="13.89"/>

--- other edge definitions

</edges>
```

2.2.1.2 VEHICLES AND ROUTES

A vehicle in SUMO consists of three basic parts: vehicle type (describing the vehicle's physical properties), vehicle route (a complete list of connected edges between the vehicle's origin and its destination) and the vehicle itself, attaching to it a unique id. Other variables exist that define the vehicles' property or behaviour e.g. the lane it will take along its route, vehicle colour, departure and arrival time, etc. The definition file for a vehicle and its route in SUMO is also done in XML. Below shows a snippet of XML code defining a vehicular route:

```
<routes>
<vType id="type1" accel="0.8" decel="4.5" sigma="0.5" length="5"
maxSpeed="70"/>

<route id="route0" color="1,1,0" edges="beg middle end rend"/>

<vehicle id="0" type="type1" route="route0" depart="0" color="1,0,0"/>

<vehicle id="1" type="type1" route="route0" depart="0" color="0,1,0"/>

</routes>
```

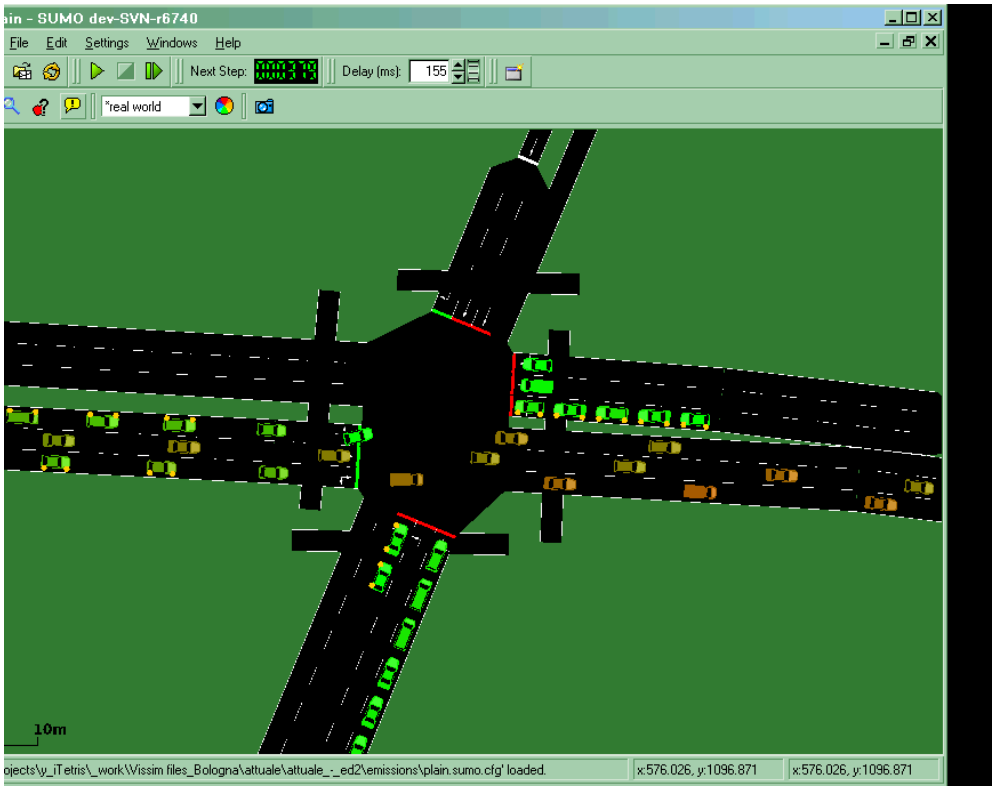


Figure 2.6: Screenshot of the SUMO-GUI colouring Vehicles by their CO2 Emission

Two versions of the traffic simulation exist, namely the SUMO and the SUMO-GUI application. SUMO application is a pure command-line application for efficient batch simulation. The application SUMO-GUI offers a graphical user interface (GUI) rendering the simulation network and vehicles using openGL. The visualization can be customized in many ways, i.e., to visualize speeds, waiting times and to track individual vehicles. The GUI also offers some possibilities to interact with the scenario, e.g. by switching between prepared traffic signal programs, changing reroute following grades, etc. Figure 2.6 shows a single

intersection simulated in SUMO-GUI. The GUI application offers all features the command line SUMO version supports (Krajzewicz et al., 2012).

2.2.2 VISUAL TRAFFIC SIMULATION

This project is a Masters thesis carried out and written by Thomas Fotherby in 2002. The work is aimed at providing a visualization of a user-defined traffic flow and is capable of comparing how different road infrastructures influence traffic systems. The project consists of two application components, which we will analyse briefly, namely Road Network Designer and the Visual Simulation.

2.2.2.1 ROAD NETWORK DESIGNER

This is the application for designing the road network, which allows a user to quickly design simple schematic road diagrams/networks to scale. This road, drawn on an initially blank canvass, is imagined as a sub-section of a larger network. Vehicles enter and exit the scene at roads which have been drawn to connect to the edge of the canvass. These roads are tagged as “input” and “output” roads. According to the author, for the application to be realistic and produce useful results, a user must be able to specify the traffic data that the simulator will use. For example, a road input must be specified with a “busyness” variable that will determine how much traffic enters the network at that particular input.

In a valid road network all road sections must end at either a junction or the edge of the canvass. Once a road network design has been created a user should be able to save it to a file and should be able to load other designs. A user can also specify an image to use as a backdrop to a road design. The file format for saving (and loading) the created road network design is in XML, with objects specified as XML Elements (tags such as <Road></Road>) and the objects' data stored as XML Attributes inside the object tag (e.g. <Road start = "x, y">).

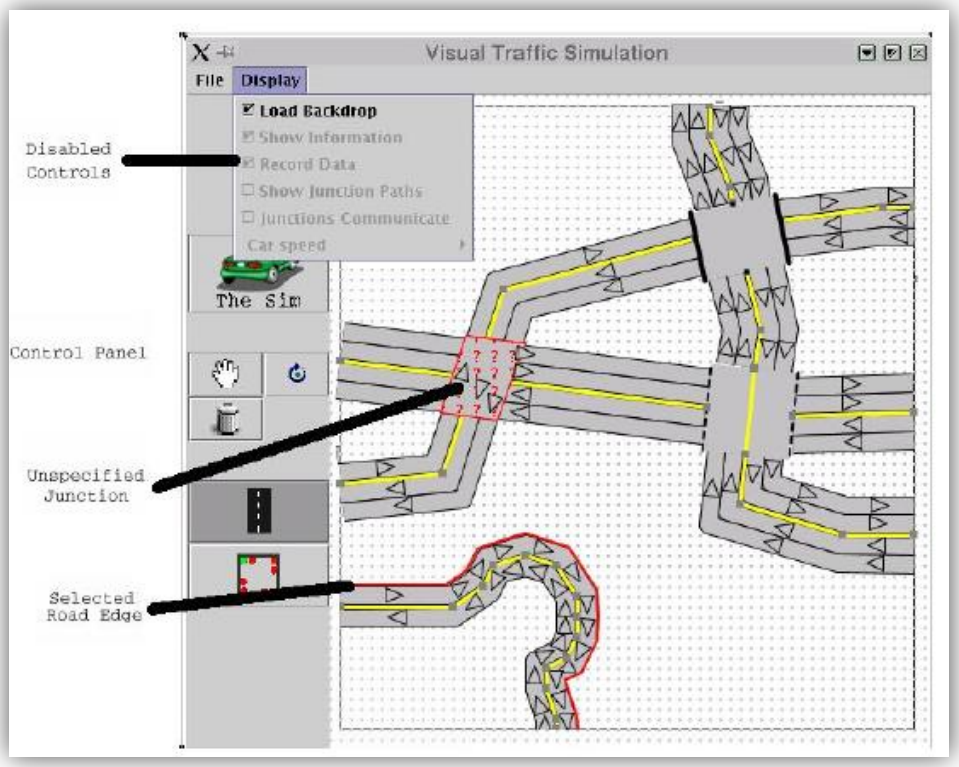


Figure 2.7: Screenshot of the Road Designer

2.2.2.2 VISUAL SIMULATION

This component presents animated graphics with drawn-to-scale vehicles moving through the geometry of the system. The animated traffic is generated and controlled according to the statistics specified by the user for each component of the design.

Every detail of the model must be specified to get the required behaviour of traffic in the road network. This implies that the vehicles on the road do not take independent decisions; rather

they follow certain rules that have been specified or put in place by the modeller. For instance, vehicles obey a specified speed limit which is tagged their “top speed”. Also, when cars enter an input road they enter at their “top speed” at positions and times according to a set traffic model specified by the user. If a car cannot enter the network because there are other cars blocking the way it should queue at the input.

The most important part of the simulation is the results it generates. Information is available, when requested, for the following:

- The current frame number of the simulation
- How many cars have entered the road network
- How many cars have exited the road network
- How many cars are currently on the road network
- The average speed of all the cars currently on the screen
- A measure of congestion (e.g. percentage of road surface covered by vehicles) for the current frame.

Also, a user is able to request a graph of the current road network. The graph should show the simulation’s throughput, average speed and congestion for a particular time period. These results will allow the level of service of a particular road network to be deduced (Fotherby, 2002). Figure 2.8 is a screenshot of the running simulation displaying current simulation frame number, total cars that have entered the simulation, total cars that have exited the simulation, total number of cars that are currently on the screen, average speed of cars and average congestion of roads.

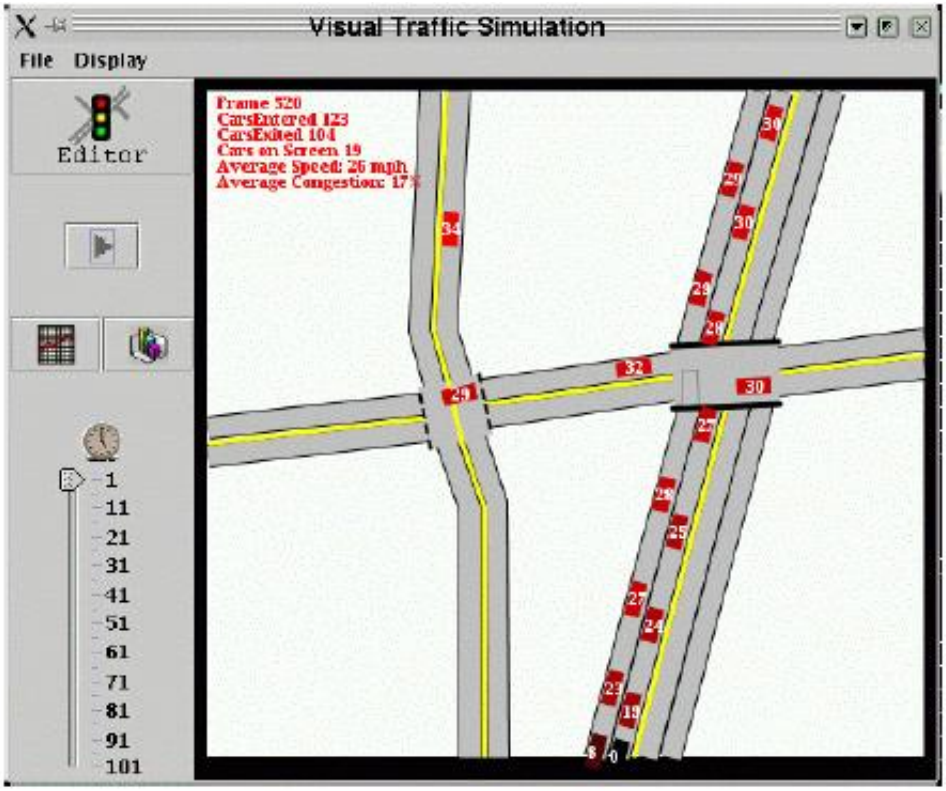


Figure 2.8: Screenshot of the Running Simulation

2.2.3 3-D VISUALIZATION FOR MICROSCOPIC DATA SOURCES

Authored by Wenger A. et al (2013), this work birthed the creation of a tool for generic, realistic 3D visualization based only the simulation geometry, vehicle trajectories and (optionally) vehicular communication data encoded in XML. The vehicles can be colour-coded, viewed from above or from a driver's perspective, or alternatively in a traditional 2-D form. Though based on microscopic traffic model, the research carried out by this team sought to bridge two major lapses of existing visualizers observed in their investigation. They introduced:

- 3D view to real-world objects and infrastructure e.g. buildings, vegetation, etc. to their visualization to present a realistic feel to the simulation
- Vehicular communication which can be seen when visualizing the simulation

They achieved this by developing a module for the vtSim simulation framework called vtSim.VIEW, which visualizes any road network, set of vehicle trajectories upon it and communicates between entities. The resulting application is a visualizer that is independent of the simulation (the main vtSim software). All the application requires is the output data from the simulation describing the road network and vehicle trajectories. The data source for the visualization is in XML format.

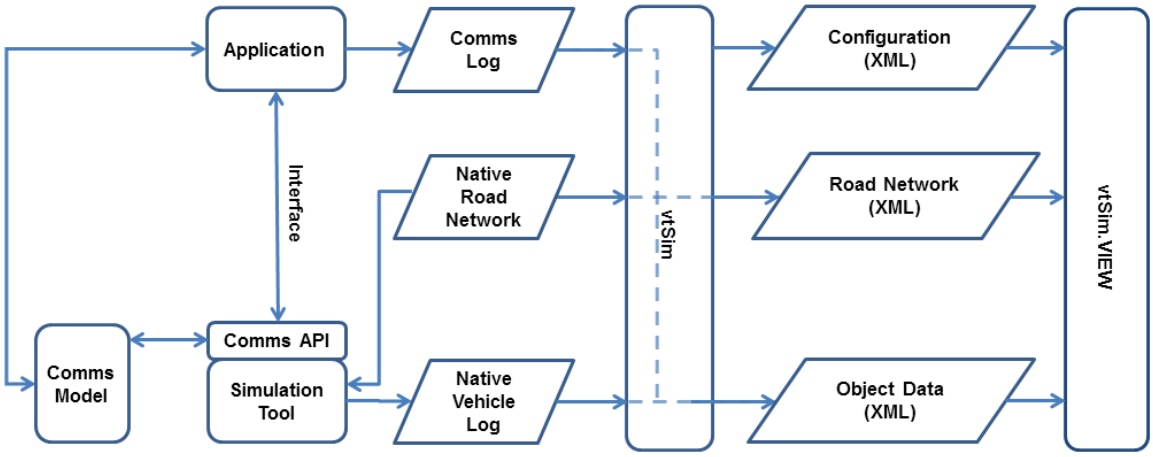


Figure 2.9: Visualization Workflow for vtSim.VIEW

The network used for simulation along with the simulation vehicle trajectories log and communications log from the simulated application are prepared as XML files for the vtSim.VIEW visualization tool. This can either be done by the user as part of their results-handling process or achieved (currently for VISSIM) using vtSim.

The minimum information to see moving vehicles are vehicle definitions with timestep entries describing their locations. In addition, communication messages and vehicle statuses can be provided, referenced to relevant timestep entries. An example of vehicle data is structured below.

```
<vehicles>
        <vehicle id="29" type="LKW">
                <timestep LateralPosRel="0.5" LinkPos="927.10"
                LinkPosRel="0.97" X="25.62" Y="-89.61" acceleration="1.42"
                lane="2" link="200733801" speed="24.67" time="300.0"
                timeGap="80.22">
        <warning distanceToEvent="9.7" type="congestion" />
        <status type="warned" />
        </timestep>
        ... further timesteps ...
        </vehicle>
... further vehicles ...
</vehicles>
```

2.2.3.1 TRAFFIC VISUALIZATION

The conversion from the network representation to a (currently motorway-specific) 3D graphic is performed by first building a wireframe model based on the network description. Connections to other links and the number of lanes are taken into account. Figure 2.10 shows the construction of a road section schematically, while Figure 2.11 shows the visualization of communication data for a vehicular communication application in a research project which used the application (Wenger et al., 2013).

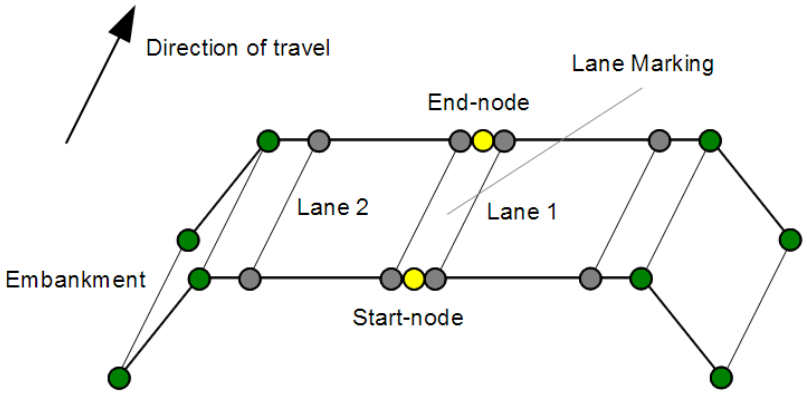


Figure 2.10: Schematic Diagram of Road Link Construction

Figure 2.11: Vehicles Receiving Warnings about Traffic Jam

The edges of roads are augmented with a hard shoulder and curbing. Empty land is shown as grass featuring undulating hills and the sky is textured. Vehicles are shown with detailed vehicle models. The size of the network displayed by vtSim.VIEW is only limited by system memory. Efficient culling techniques in the jMonkeyEngine compute only the geometry which is actually visible on the screen. The software is currently being extended to handle urban road networks and scenery, traffic lights, pedestrians, cyclists and roads that cross over one another (e.g. motorway junctions).

2.3 PROJECT PROPOSAL

Following the projects reviewed above, our work proposes an independent animation tool that has the following advantages:

- Generic: Can take vehicle trajectory data source from any simulator, irrespective of the traffic model (micro-, meso-, macroscopic) considered, and present its visualization.

- Realistic: We are proposing to use Google Maps, in its dynamic form, as the backdrop of our visualizer. The simulation can be viewed on any target simulated road.
- User Friendly: The use of Google Maps (in its dynamic form) is to solve the problem of designing a road from scratch using a road editor, or writing code to simulate a virtual road, as opposed to the method(s) that obtain in the related works outlined above.

The data source for our work will also be in XML format, as they can be efficiently read, exchanged and their format defined in a standardized way using XML schema.

Since we are designing a software, we would be making use of the RAD (Rapid Application Development) approach of software development methodology. The approach focuses on using the prototype (light version of the final product) as a basis for project kick-off and for continuous and iterative design evolution with the user. Also, the few critical factors for this consideration include:

i. Limited delivery time
ii. Individual developer and not team
iii. Good understanding of the requirements needing no repeated and frequent client engagement.

CHAPTER 3
DESIGN METHODOLOGY

3.1 OVERVIEW

As mentioned in the previous chapter, the main objective of this project is to design an animation tool to visualize traffic simulation traces/result on Google Maps. Google Maps is a web-based service that provides detailed information about geographical regions and sites around the world. In this chapter, we discuss in detail the concept of our design methodology.

3.2 DESIGN CONCEPT

For our animation software, we propose to develop a desktop application that implements a web-based user interface. This is because our software should accept input and provide output by generating a Google Map location. This output will be transmitted via the internet and viewed by the user using a web browser program. To achieve this, the application will be designed with JavaFx, which will include an html file. Also, we will need an XML file that will store the trace data obtained from the traffic simulation. Fig. 3.2 shows the package diagram depicting the relationship between these components. Also, we will elaborate, briefly, on these components so as to give an idea on why we want to use them to achieve our goal.

3.2.1 SOFTWARE COMPONENTS

JavaFX: JavaFX is a software platform for creating and delivering desktop applications, as well as rich internet applications (RIAs) that can run across a wide variety of devices. It has a user interface component, the **JavaFX embedded browser**, which provides a web viewer and full browsing functionality through its API. The embedded browser enables you to perform the following tasks in your JavaFX applications:

- Render HTML content from local and remote URLs
- Obtain web history
- Execute JavaScript commands
- Perform upcalls from JavaScript to JavaFX
- Manage web pop-up windows

- Apply effects to the embedded browser.

The WebView and the WebEngine are classes from the javafx.scene.web package. The WebView is an extension of the Node class. It encapsulates a WebEngine object, incorporates HTML content into an application's scene and provides properties and methods to apply effects and transformations. The WebEngine on the other hand is a non-visual object capable of managing one web page at a time. It supports user interaction such as navigating links and submitting HTML forms, although it does not interact with users directly. The WebEngine class handles one web page at a time. It supports the basic browsing features of loading HTML content and accessing the DOM (Document Object Model) as well as executing JavaScript commands.

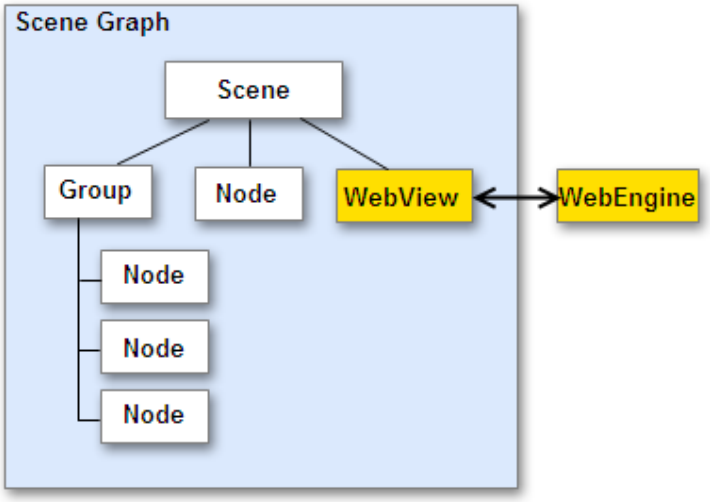


Figure 3.1: The Architecture of an Embedded Browser

HTML: Because everything on the web requires HTML, we need to create an HTML file so as to be able to load and configure our map using JavaScript. To load a map, we must add two fundamental pieces of JavaScript to our html code. The first loads the Google Maps JavaScript API while the second creates and configures the map. The Google Maps JavaScript API is a powerful, popular mapping API. It allows a user customize maps, and the information on maps. It is used to add maps to a website, or web or mobile application, and provides a wide range of services and utilities for data visualization, map manipulation, directions and more.

XML: XML (Extensible Markup Language) is an exciting development in web technology. It is used to outsource data. Rather than integrating your data into the HTML document, it is stored in separate XML files. Since XML stores data in plain text format, the storage is platform independent and data can be exported, imported or simply moved much more easily. XML allows a user to create customized tags according to need.

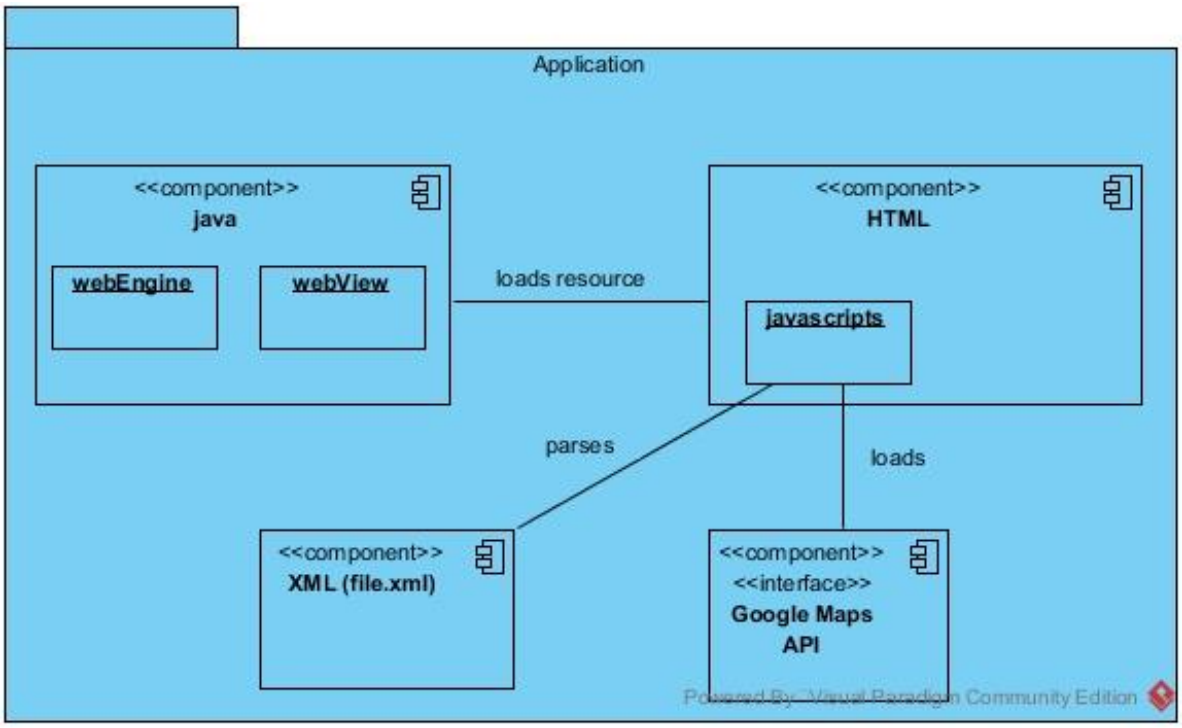


Figure 3.2: Package Diagram of the System Structure

CHAPTER 4
IMPLEMENTATION AND TESTING

4.1 INTRODUCTION

In this section, we will look at each component discussed in the previous chapter and translate them to their specific code formats. We will also mention/explain the functionalities of some fundamental elements of the component where necessary.

As mentioned in the previous chapter, our application will be designed with JavaFx. The Java IDE used is NetBeans IDE 8.1. Our source package will comprise of the java file that renders the GUI, the html code, containing scripts (JavaScript) that transmits the browser (Google Map), and an XML file used for specifying and storing the data that will be used for the visualization of the traffic simulation.

4.2 CODE TRANSLATION

4.2.1 JAVA CODE

The Java code consists of only one class, which extends the Application class (the entry point for JavaFx applications is the Application class). This class contains a start() method which is called after the system is ready for the application to begin running.

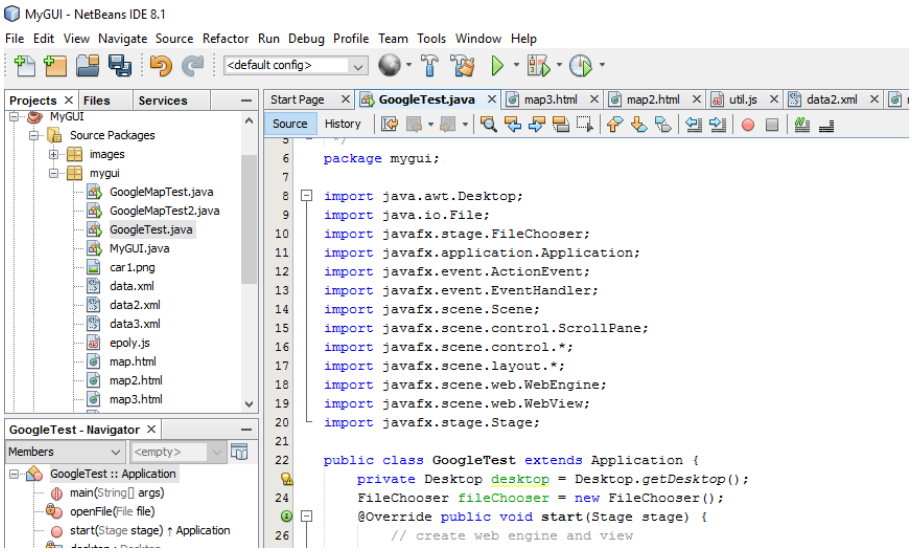


Figure 4.1: JavaFx Code Snippet

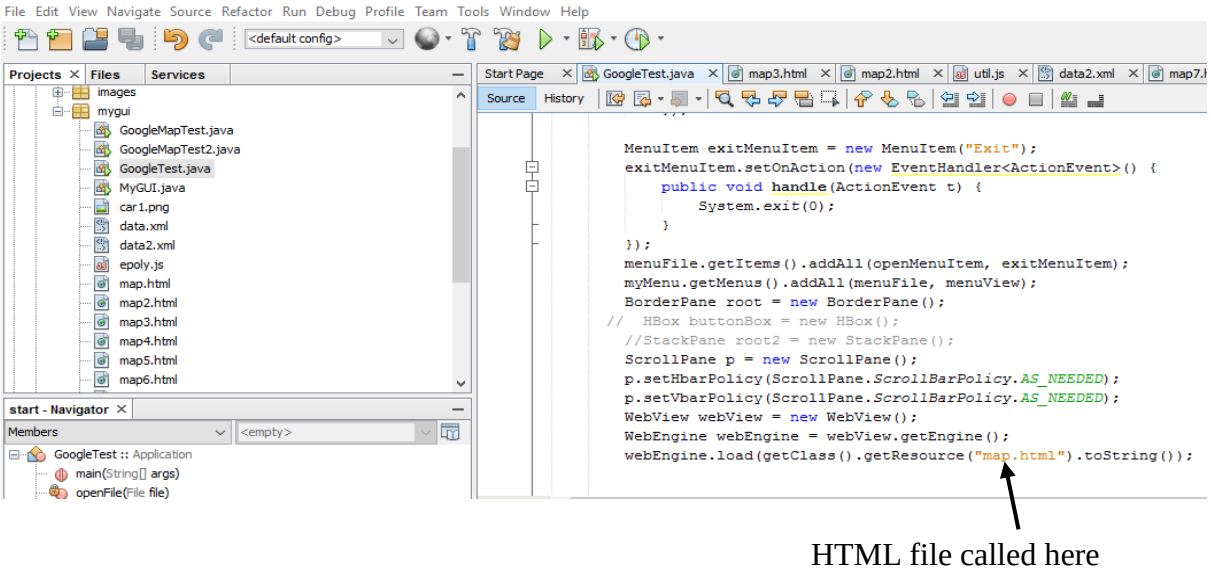


Figure 4.2: 12HTML file load via the JavaFx WebEngine Object

4.2.2 THE HTML FILE

The bulk of the code that runs our software lies here. It contains the JavaScript scripts that will render the Google Map. The map is placed and appears in a div tag, which serves as its container.

```
<?xml version="1.0" encoding="UTF-8"?>
<vehicles>
   <timeStamp id = "0" time = "0">
      <car id = "1" color = "red" lat="9.053224488711281" lng="7.481206655502319"/>
      <car id = "01" color = "blue" lat="9.053637700595743" lng="7.481871843338013"/>
   </timeStamp>
   <timeStamp id = "1" time = "2000">
      <car id = "2" color = "blue" lat="9.053224488711281" lng="7.481206655502319"/>
      <car id = "1" color = "red" lat="9.053637700595743" lng="7.481871843338013"/>
      <car id = "01" color = "black" lat="9.054019126529168" lng="7.482622861862183"/>
   </timeStamp>
   <timeStamp id = "2" time = "4000">
      <car id = "3" color = "black" lat="9.053224488711281" lng="7.481206655502319"/>
      <car id = "2" color = "blue" lat="9.053637700595743" lng="7.481871843338013"/>
      <car id = "1" color = "red" lat="9.054019126529168" lng="7.482622861862183"/>
      <car id = "01" color = "green" lat="9.054739596632766" lng="7.483727931976318"/>
   </timeStamp>
```

```
    <timeStamp id = "3" time = "6000">
        <car id = "4" color = "green" lat="9.053224488711281" lng="7.481206655502319"/>
        <car id = "3" color = "black" lat="9.053637700595743" lng="7.481871843338013"/>
        <car id = "2" color = "blue" lat="9.054019126529168" lng="7.482622861862183"/>
        <car id = "1" color = "red" lat="9.054739596632766" lng="7.483727931976318"/>
        <car id = "01" color = "yellow" lat="9.054665430659337" lng="7.4839746952056885"/>
    </timeStamp>
    <timeStamp id = "4" time = "8000">
        <car id = "5" color = "yellow" lat="9.053224488711281" lng="7.481206655502319"/>
        <car id = "4" color = "green" lat="9.053637700595743" lng="7.481871843338013"/>
        …
    </timeStamp>
</vehicles>
```

Figure 4.3: Basic Code to Load the Google Maps API and Create a Map

There are a number of options that can be used to define the map to be rendered, but the two required ones can be seen in the code above. They are:

- **center** is a Google Maps `LatLngLiteral` object that tells the API where to centre the map.
- **zoom** is a number between 0 (farthest) and 22 that sets the zoom level of the map.

We also note that, to be able to execute the JavaScript function, a callback has been added to the script tag that loads the API. The callback waits until the <script> finishes loading before calling the initMap function (Calling initMap before the API has finished loading will cause problems, since the methods and functions may not have been initialized yet).

The code above, when run, will produce this kind of map.

Figure 4.4: Google Map Example Loaded from JavaScript Using its API

4.2.2.1 GOOGLE MAPS API MAP CLASS

This class has a constructor, several methods, properties and events. The constructor is what creates a new map inside of the given html container i.e. the div element. One example of the map property is the mapTypeId. The **mapTypeId** property specifies the map type to display. The following map types are supported:

- ROADMAP (normal, default 2D map)
- SATELLITE (photographic map)
- HYBRID (photographic map + roads and city names)
- TERRAIN (map with mountains, rivers, etc.).

JavaScript within the browser is *event driven*, meaning that JavaScript responds to interactions by generating events, and expects a program to *listen* to interesting events. There are two types of events:

- User events (such as “click” mouse events) are propagated from the DOM to the Google Maps JavaScript API. These events are separate and distinct from standard DOM events.

- MVC state change notifications reflect changes in Maps JavaScript API objects and are named using a *property_changed* convention.

Each Maps JavaScript API object exports a number of named events. Programs interested in certain events will register JavaScript **event listeners** for those events and execute code when those events are received by calling the addListener() event handler to register event handlers on the object. List of events include click, rightclick, dblclick, bounds_changed, mouseover, drag, dragend, etc. They all follow this syntax: google.maps.event.addlistener('event_name', function(){…function code to be executed when event triggers…});. Some of these events mentioned were incorporated in our design as we will see in the testing section.

4.2.3 THE XML FILE

As was stated in the previous chapter, the XML file will be the data source for our work. The content of this file will be specified to hold, mainly, our vehicle information; the car type, the time a car entered the road, car speed, and most importantly the coordinates (longitudes and latitudes) that gives the point where a car is on the map per time. To be able to load/access data from the xml file using JavaScript, the file will need to be parsed. XML parsing refers to going through XML document to access data or modify data in one way or another. There are various types of parsers that are commonly used to parse XML documents. These are:

- **DOM Parser**: parses the document by loading the complete contents of the document and creating its complete hierarchical tree in memory.
- **SAX Parser** - Parses the document on event based triggers. Does not load the complete document into the memory.
- **JDOM Parser** - Parses the document in similar fashion to DOM parser but in an easier way.
- **StAX Parser** - Parses the document in similar fashion to SAX parser but in more efficient way.
- **XPath Parser** - Parses the XML based on expression and is used extensively in conjunction with XSLT.
- **DOM4J Parser** - A Java library to parse XML, XPath and XSLT using Java Collections Framework, provides support for DOM, SAX and JAXP.

DOM parser will be used to parse our XML file for the purpose of this project. For our work, the DOM parser was defined and saved as an external script in a .js file and was called as a script source in the html file.

4.3 SYSTEM PROCESS

In this section, we show with a sequence diagram how the objects of our software interact with each other to create a functioning system. When running the animation, the user views the traffic state of a given location per time.

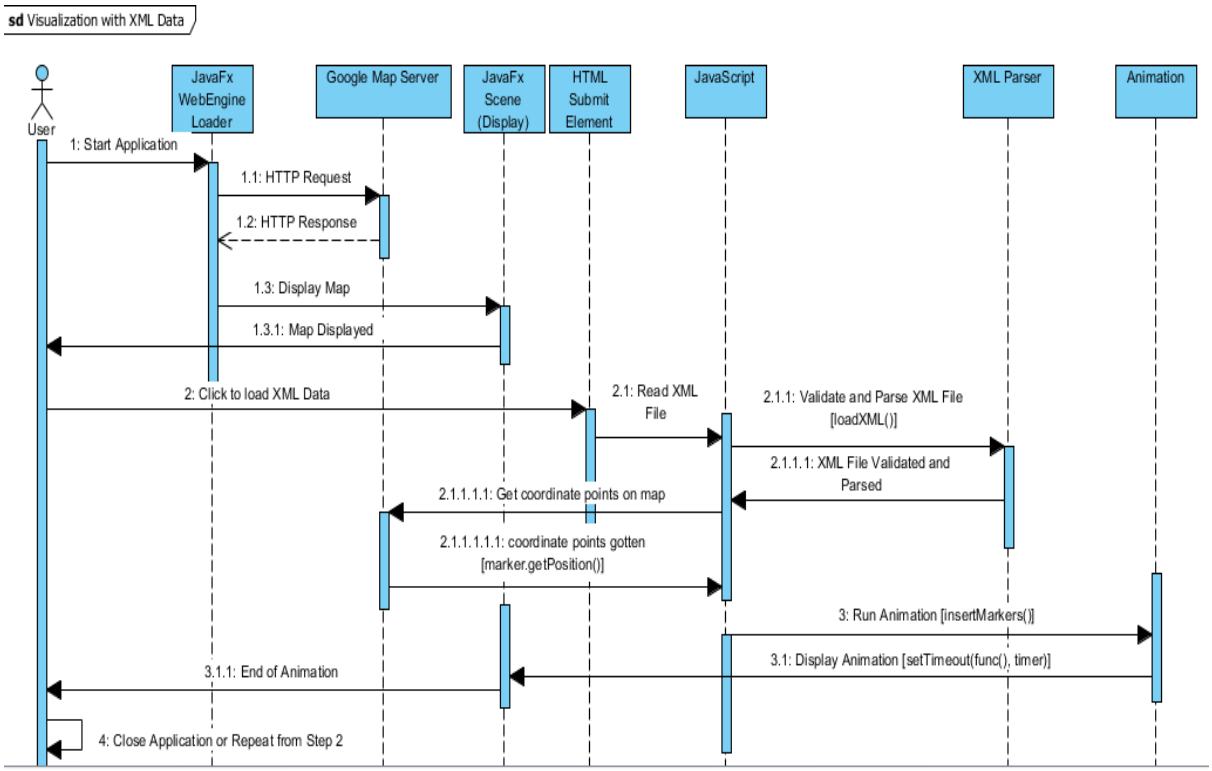


Figure 4.5: Sequence Diagram of the Animation Tool System Process with XML Data

4.3.1 ALTERNATIVE APPROACH

In the course of the development of the software, we devised an alternative approach to using the animation tool. This was made possible with the DirectionsService object of the Google Maps API. This object communicates with the Google Maps API Directions Service which receives direction requests and returns computed results.

To use directions in the Google Maps JavaScript API, create an object of type DirectionsService and call DirectionsService.route() to initiate a request to the Directions

Service, passing it a `DirectionsRequest` object literal containing the input terms and a callback method to execute upon receipt of the response. The syntax for using this object is given as:

```
directionsService.route(request, function(result, status){
   if (status === google.maps.DirectionsStatus.OK){
        directionsDisplay.setDirections(result);
}
```

The directionsDisplay is an instance of the DirectionsRenderer object, which renders the direction result that was returned from the direction request made to the DirectionService object.

Accessing the Directions Service is asynchronous, since the Google Maps API needs to make a call to an external server. For that reason, we needed to pass a callback method to execute upon completion of the request. This callback method should process the result(s). Note that the Directions Service may return more than one possible itinerary as an array of separate routes[].

The DirectionsRequest object literal contains the following basic fields:

```
{
        origin:LatLng | String | google.maps.Place,
        destination: LatLng | String | google.maps.Place,
        travelMode: TravelMode,
        ….
}
```

These fields are explained below:

- origin (*required*): specifies the start location from which to calculate directions. This value may be specified as a String (for example, "Chicago, IL"), as a LatLng value or as a google.maps.Place object.
- destination (*required*): specifies the end location to which to calculate directions. The options are the same as for the origin field described above.
- travelMode (*required*): specifies what mode of transport to use when calculating directions. These include Driving (default), Bicycling, Walking and Transit (returns public transport routes).

Although we could store the origin and destination points in an XML file as we proposed from inception, we designed this approach to read the origin and destination input dynamically by clicking on the map using the addListener(click,function(){…} event handler. As soon as the user clicks the destination point (we assume first click gives input for origin and second for destination), the DirectionService LatLng literals of these points, calculates and returns a route. We designed the software to start the animation as soon as the route is returned. We also supplied an HTML input element which reads a number from the user. This will determine the number of vehicles that traverse the road during the animation.

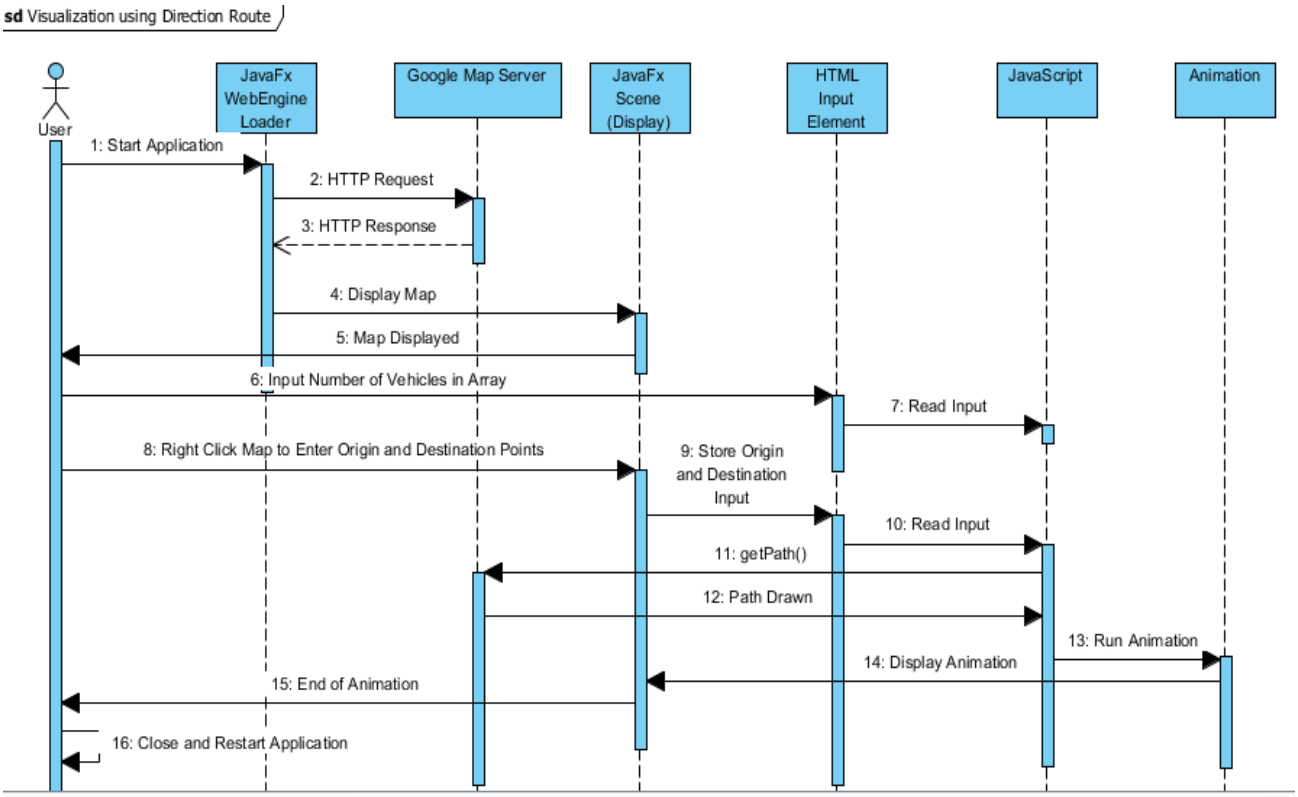


Figure 4.6: Sequence Diagram of the Animation Tool System with Direction Service

4.4 TESTING

4.4.1 FIRST APPROACH

We test our software using the first approach. In designing the approach, we have put an input field we named “Marker Positions” to enable a user get coordinate points from the map to populate the XML file. In order to get a map point, right click on the map. We show screenshots below of our test results, and then we explain its features.

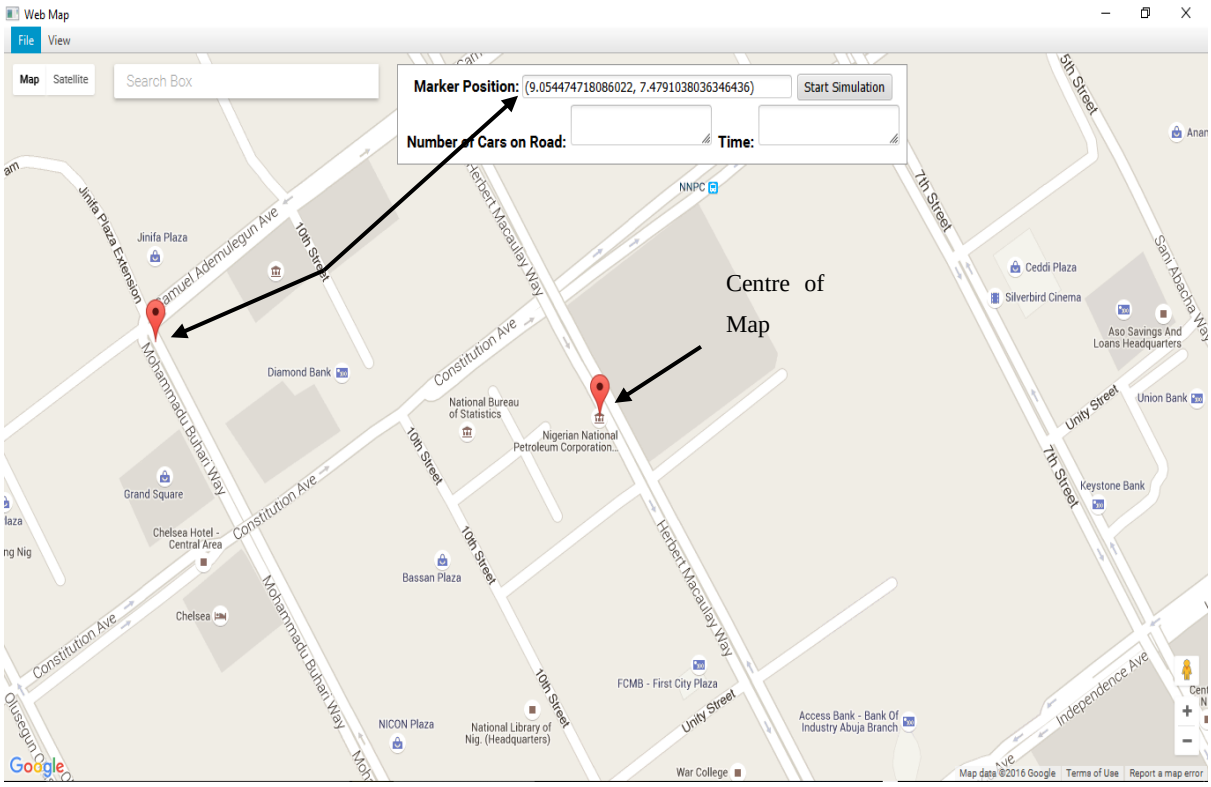


Figure 4.7: Google Map Showing Marker Position in the HTML Input Element

We recall from the previous chapter that one of the basic requirements to render a map is to supply a LatLng literal in the map options where the map will be centred. For our case study we supplied, by default in the code, a point that will centre the map in Central Business District Abuja, Nigeria. If a user needs to change the viewport, he/she will be required to use the search box field at the top left corner. This feature allows a user to type in a target location and the map will pan to the address selected in the search box.

We also incorporated the autocomplete for addresses and search terms. Autocomplete is a feature of the Places library in the Google Maps JavaScript API. You can use autocomplete to give your applications the type-ahead-search behaviour of the Google Maps search field. When a user starts typing an address, autocomplete will fill in the rest.

The text area elements labelled “Number of Cars” and “Time” outputs the number of cars on the road and at what time (in seconds) respectively, during the simulation.

Below is the XML file we will use to run our animation.

```
<?xml version="1.0" encoding="UTF-8"?>
<vehicles>
    <timeStamp id = "0" time = "0">
        <car id = "1" color = "red" lat="9.053224488711281"
lng="7.481206655502319"/>
        <car id = "01" color = "blue" lat="9.053637700595743"
lng="7.481871843338013"/>
    </timeStamp>
    <timeStamp id = "1" time = "2000">
        <car id = "2" color = "blue" lat="9.053224488711281"
lng="7.481206655502319"/>
        <car id = "1" color = "red" lat="9.053637700595743"
lng="7.481871843338013"/>
        <car id = "01" color = "black" lat="9.054019126529168"
lng="7.482622861862183"/>
    </timeStamp>
    <timeStamp id = "2" time = "4000">
        <car id = "3" color = "black" lat="9.053224488711281"
lng="7.481206655502319"/>
        <car id = "2" color = "blue" lat="9.053637700595743"
lng="7.481871843338013"/>
        <car id = "1" color = "red" lat="9.054019126529168"
lng="7.482622861862183"/>
        <car id = "01" color = "green" lat="9.054739596632766"
lng="7.483727931976318"/>
    </timeStamp>
    <timeStamp id = "3" time = "6000">
        <car id = "4" color = "green" lat="9.053224488711281"
lng="7.481206655502319"/>
        <car id = "3" color = "black" lat="9.053637700595743"
lng="7.481871843338013"/>
        <car id = "2" color = "blue" lat="9.054019126529168"
lng="7.482622861862183"/>
        <car id = "1" color = "red" lat="9.054739596632766"
lng="7.483727931976318"/>
        <car id = "01" color = "yellow" lat="9.054665430659337"
lng="7.4839746952056885"/>
    </timeStamp>
    <timeStamp id = "4" time = "8000">
        <car id = "5" color = "yellow" lat="9.053224488711281"
lng="7.481206655502319"/>
        <car id = "4" color = "green" lat="9.053637700595743"
lng="7.481871843338013"/>
        …
    </timeStamp>
</vehicles>
```

Figure 4.8: XML Code Specified To Hold Vehicle Information

For now, we only specified an XML document that takes into account only the car id, car colour and its position on the road at a point in time within the simulation time range. The car with id number “01” is not displayed on the road. We have put in that element to help us get and set properly the car direction on the road. This was achieved using the google.maps.geometry.spherical.computeHeading(latlng,latlng2) constructor. It takes two

arguments which are the current position of a vehicle and the next position where it will be set. This will help flip the vehicle in the right direction.

After we ensure our XML document complies with this specification, save the file with the name “data.xml” and save it into the same source folder of the html code. When that is ensured, we click on the button “Start Simulation” to view our animation.

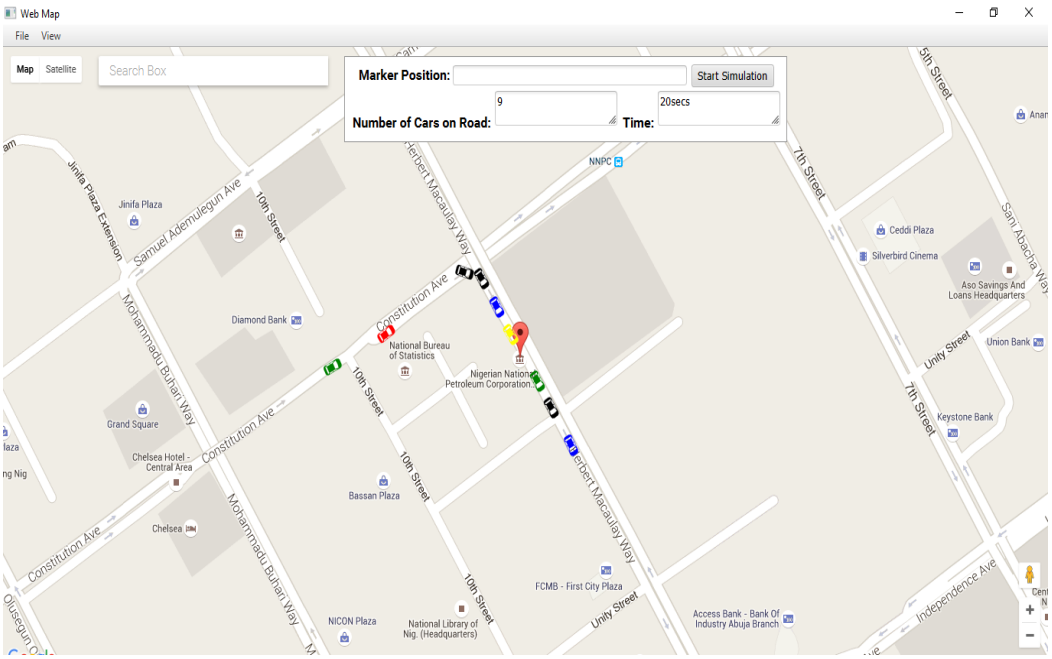


Figure 4.9: Animation Using Vehicle Information in XML File

4.4.2 ALTERNATIVE APPROACH

To use this approach, the first thing we do, provided the map is centred on our desired target, is to enter a number in the input field which will determine the number of cars that will traverse the road during the simulation. This number must be supplied before we go ahead to supply the origin and destination points by right-clicking on the map. This will ensure that, from our design, the direction route will be calculated and the animation function will be called. Whenever the rightclick event is fired, a marker is inserted at the point that was clicked. A marker identifies a location on a map.

We also labelled our marker alphabetically for easy identification. Based on our earlier assumption that the first click to get a point is our origin and the second to be our destination point, therefore the labelling from “A” and every other letter after it indicates our origin

position while the labelling from “B” and every other letter after it indicates the corresponding destination position.

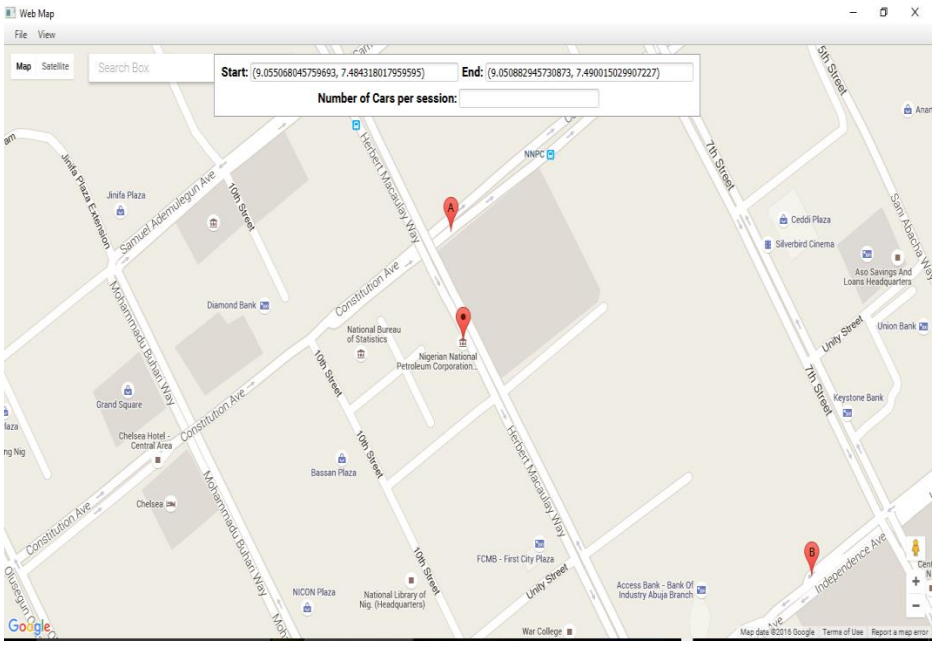
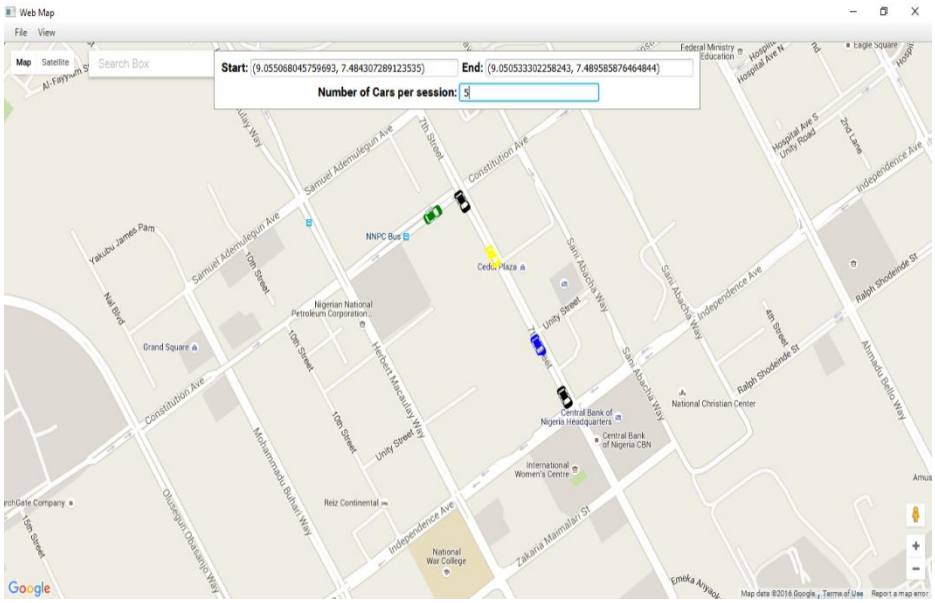


Figure 4.10: Google Map Showing Labelled Markers



We run the animation to obtain the screenshot below:

Figure 4.11: Animation for the Direction Service Implementation

When sending a directions request to the DirectionsService, you receive a response consisting of a status code, and a result, which is a DirectionsResult object. The DirectionsResult is an object literal with a routes[] fields which contains an array of DirectionsRoute objects. Each route indicates a way to get from the origin to the destination provided in the DirectionsRequest. Generally, only one route is returned for any given request, unless the request's provideRouteAlternatives field is set to true, in which, multiple routes may be returned.

The DirectionsRenderer handles the display of a polyline between the indicated locations. To draw a line on your map, use a polyline. The Polyline class defines a linear overlay of connected line segments on the map. A Polyline object consists of an array of LatLng locations, and creates a series of line segments that connect those locations in an ordered sequence.

CHAPTER 5
GENERAL CONCLUSION

5.1 OBSERVATION

As we conclude our work, we can say that we were able to achieve our aim to a large extent because we were able to achieve dynamic interactions with the Google Map via its API. For what it is worth, the difficulty of building a target road from scratch with an editor was tremendously reduced. This sets/adds to the framework of knowledge in designing/developing a traffic animation tool which developers can use and build upon to achieve a state of the art design.

5.2 ASSUMPTIONS AND LIMITATIONS

Due to the constraint for delivery time that we had to design this application, we made certain assumptions to enable us realize the workability of the idea we had in mind. After our realization, we could not give enough time to implement other algorithms that govern road traffic movement. One of such notable assumptions that we made was the speed of the car.

We assumed that the vehicles were moving at the same speed and that vehicles were generated to enter the road at a fixed time interval. By doing so, we were able to display an animation that seemed to comply with the car-following and gap acceptance algorithm of traffic simulations. Although we tried to define different times at which the cars are generated and a different speed for each car that enters the road by increasing/decreasing the animation time it will take the car to get to its destination (since speed is distance/time), we could not come up with a solution that will ensure that the car following will reduce its speed when it is close enough to the car in front of it, even when we tried to use a delay function to check to see if the car will wait when it is close to the leading car. Rather, when the car behind approaches the one in front of it, it goes over the car in a bid to overtake it.

One other limitation we had was that we could not implement the lane-changing algorithm. This is peculiar to the second approach of our design. We realize that when the Directions Service calculates and returns the route, it returns a route that is centred on the road, not giving room for having separate lanes. Also, due to the dynamic style we adopted for this

approach, we could not get two cars to enter road at the same time, side by side – one will be on top of the other

5.3 FUTURE WORK

Research can go into the limitations we outlined above to see how the issue of dynamic car-following, gap acceptance and lane changing can be solved. One other solution to consider is how to incorporate traffic light phases in the design to give an extended realistic feel, especially in the second approach of our work.

REFERENCES

(Barcelo, 2010) Fundamentals of Traffic Simulation, Springer Book, edited by James Barcelo, Preface page (pp vii) & pp 273

(Payne, 1979) Payne Harold, 1979. FREFLO, A Macroscopic Simulation Model of Freeway Traffic

(Messmer et al, 1990-9) Messmer A. & Papageorgiou M., (1990-9). METANET: A Macroscopic Simulation Program for Motorway Networks, pp 466-470

(Shekar et al, 1997) Shekar S., C.T. Lu, R. Liu, & C. Zhou, University of Minnesota, (1999). CubeView: A System for Traffic Data Visualization, page 1

https://sourceforge.net/projects/transims-metro

(Wenger et al, 2013) Wenger A., & Matthew F., (2013). 3D Visualization for microscopic traffic data sources, pp. 81

(Sharon et al, 2001) Sharon A.B. & Lei Y. (2001). An Evaluation of Traffic Simulation Models for Supporting ITS Development, pp 21-24.

(Matthew et al, 2007) Matthew B., Kathy D., David K., Kevin M., Euneka R., Duane W., Adam S., John M., & Mark H., (2007). A Guide to Documenting VISSIM-Based Microscopic Traffic Simulation Models, pp 12.

(TRC, 2015) Transportation Research Circular, 2014. Traffic and Transportation Simulation, Looking Back and Looking Ahead, Traffic Flow Theory 50th Anniversary, pp 9, 12.

(Kessels, 2013) Femke Van Wageningen-Kessels (2013). Multi-Class Continuum Traffic Flow Models: Analysis and Simulation Methods, pp 44.

www.wisdot.info

https://en.wikipedia.org/wiki/James_Lighthill, accessed 13th April, 2016

https://en.wikipedia.org/wiki/Macroscopic_Traffic, accessed 13th April, 2016

(Brackstone et al, 1998) Brackstone M., & McDonald M. (1998). Car-Following: A Historical Review, pp 182.

(Fotherby, 2002) Thomas Fotherby (2002). Visual Traffic Simulation, an MEng Thesis Report, pp 17.

(Charles M., 2001) Charles M. Macal (2001). Simulation and Visualization, pp 90

(Matti, 1999) Matti Pursula, (1999). Simulation of Traffic Systems – An Overview. Retrieved from http://publish.uwo.ca/~jmalczew/gida_5/Pursula/Pursula.html

(Kotusevski et al., 2009) Kotusevski G., & Hawick K.A., (2009). A Review of Traffic Simulation Software, pp 39

(Karnadi et al., 2007) Karnadi, F. K., Mo, Z. H., & Lan, K. (2007). Rapid Generation of Realistic Mobility Models for VANET (Vehicular Ad-hoc Network), 2508–2513.

(Wenger et al., 1987) Wenger, A., Fullerton, M., & Baur, M. (n.d.). 3D visualization for microscopic traffic data sources. *Mediatum.Ub.Tum.De*, 80–87. Retrieved from http://mediatum.ub.tum.de/doc/1191987/1191987.pdf

Treiber, M. (2009). Microsimulation of Road Traffic Applet. Fig. 2.3 source retrieved from http://www.traffic-simulation.de, accessed March 12, 2016

Quadstone Paramics Website (2009), Fig. 2.4 source retrieved from http://www.paramics-online.com, accessed 13th April, 2016

(Krajzewicz et al., 2012) Krajzewicz D., Erdmann J., Behrisch M., & Bieker L.,(2012). Recent Development and Applications of SUMO – Simulation of Urban Mobility, a paper presented at the International Journal on Advances in Systems and Measurement, vol. 5, no 3-4.

https://en.wikipedia.org/wiki/JavaFx, accessed 12th May, 2016

www.tutorialspoint.com/java-xml , accessed 12th May, 2016

https://developers.google.com/maps/documentation/javascript/tutorials/adding-a-google-map#try_it_out , accessed 12th April, 2016

docs.oracle.com/javafx/2/overview/jfxpub-overview.htm, accessed 12th May, 2016

http://creately.com/blog/diagrams/uml-diagram-types-examples/ accessed 25th May, 2016

https://developers.google.com/maps/documentation/javascript/directions#DisplayingResults, accessed 12th April, 2016

https://developers.google.com/maps/documentation/javascript/markers, accessed 12th April, 2016

https://developers.google.com/maps/documentation/javascript/shapes#polylines, accessed 12th April, 2016

https://developers.google.com/maps/documentation/javascript/reference#Map, accessed 12th April, 2016