A Cloud-Based Visual Simulation Environment for Traffic Networks

Sait Tuna Onder

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

Osman Balci, Chair
Denis Gracanin
Na Meng

May 8, 2018

Blacksburg, Virginia

Keywords and phrases: Cloud-based visual simulation, integrated development environment, modeling and simulation, web-based visual simulation

A Cloud-Based Visual Simulation Environment for Traffic Networks

Sait Tuna Onder

ABSTRACT

Cloud-based Integrated Development Environments (IDEs) are highly complex systems compared to stand-alone IDEs that are installed on client devices. Today, the visual simulation environments developed as services on the cloud can offer similar features as client-based IDEs thanks to the advancements to the cloud technologies. However, most of the existing visual simulation tools are developed for client-based systems. Moving towards the cloud for visual simulation environments can provide better collaboration for simulation developers, easy access to the software, and less client hardware dependency. Proper guidance for the development of visual simulation tools can help researchers to develop their tools as a service on the cloud. This thesis presents a **c**loud-based visu**a**l simulatio**n** en**v**ironment for tr**a**ffic network**s** (CANVAS), providing a framework that tackles challenges on the cloud-based visual simulation tools. CANVAS offers a set of tools for the composition and visualization of simulation models for the traffic network problem domain. CANVAS uses an asynchronous visualization protocol with efficient resource utilization on the server, enabling concurrent usage of the IDE. The simulation is executed on the server while the visualization is processed on the client-device within web browsers enabling execution-heavy simulations to thin clients. The component-based architecture of CANVAS offers a fully decoupled system that provides easier development and maintenance. The architecture can be used for the development of other cloud-based visual simulation IDEs. The CANVAS design and asynchronous visualization protocol show that advanced visualization capabilities can be provided to the client without depending on the client hardware.

A Cloud-Based Visual Simulation Environment for Traffic Networks

Sait Tuna Onder

GENERAL AUDIENCE ABSTRACT

Doing things "in the cloud" has become ubiquitous. The term "in the cloud" implies that a software application runs on a server computer somewhere in the world and a user with a web browser uses it over the Internet on a computer such as desktop or laptop. This thesis addresses the problem of how to create and execute a visual simulation of a system all "in the cloud". We developed a system called **c**loud-based visu**a**l simulatio**n** en**v**ironment for tr**a**ffic network**s** (CANVAS). We selected traffic networks as an example problem domain to illustrate the capabilities of CANVAS. A person interested in creating and executing a visual simulation of a traffic network "in the cloud" can use CANVAS. Using a web browser on an Internet-connected computer, the user can develop and execute a visual simulation of a traffic network with the tools made available in CANVAS.

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor Dr. Osman Balci for his professional guidance. I have experienced my most productive years by greatly improving my skills as a software engineer thanks to him. Most importantly, I am extremely grateful for his personal support during my education at Virginia Tech.

I also would like to thank my committee members Dr. Denis Gracanin and Dr. Na Meng for their time and guidance and my good friend Ayaan Kazerouni for his support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACROYNMS

2D	Two Dimensional
3D	Three Dimensional
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CANVAS	Cloud-based visuAl simulatioN enVironment for trAffic networkS
CBS	Cloud-Based Simulation
CDI	Context-Dependency Injection
CF	Conceptual Framework
CGI	Common Gateway Interface
CSS	Cascading Style Sheets
EJB	Enterprise Java Bean
EL	Expression Language
FOL	Future Objects List
HLA	High Level Architecture
HTML	HyperText Markup Language
IDE	Integrated Development Environment
Java EE	Java platform, Enterprise Edition
JDBC	Java Database Connectivity
JSF	JavaServer Faces
JSON	JavaScript Object Notation
M&S	Modeling and Simulation
PHP	Personal Home Page
RDBMS	Relational Database Management System
RMI	Remote Method Invocation
SaaP	Software as a Product
SaaS	Software as a Service
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
VRML	Virtual Reality Modeling Language
WBS	Web-Based Simulation
WebGL	A 3D graphics API based on OpenGL ES specification
WS	WebSockets
X3D	A royalty-free ISO standard for declaratively representing 3D computer graphics
XML	Extensible Markup Language

Chapter 1: Problem Definition and Overview

1.1 Introduction

Technological advancements in cloud computing have deeply affected a broad range of software systems. Traditional software systems follow the Software as a Product (SaaP) paradigm that allows users to install a purchased software onto a single device. On the other hand, much of the software we use today has been transitioning to the Software as a Service (SaaS) paradigm. Under this paradigm, applications are provided to users via browsers by being hosted on at least one server computer.

Software developers have been adopting SaaS over SaaP thanks to its advantages such as multi-platform support, easy maintenance, user-transparent updates, easy scalability, quick access to the software from anywhere, and less dependence on client machine hardware. Improvements to network infrastructure and increasing computing capabilities of server computers have been accelerating this change.

The idea of utilizing the SaaS paradigm has been creating new challenges for simulation developers. Discussions on the concept of Web-Based Simulation (WBS) started by Fishwick [1996]. An increasing amount of time has been devoted to WBS by researchers since then. The first conference focused on Web-based modeling and simulation was held in 1998 [Byrne *et al.* 2010]. In spite of over a decade of research in this area, there is still a lot to do in WBS, or Cloud-Based Simulation (CBS) with the recent terminology, to catch up with fast-paced developments in cloud software technologies.

Today, we can develop modeling and simulation (M&S) tools that can be used over a web browser from all kinds of client devices. Multiple advancements have allowed us to be able to implement such tools. First, improvements to visualization capabilities of web browsers reduced the need to install client software that provides powerful visualization. Second, recent developments in cloud computing started to provide cheap and reliable computing to cloud service subscribers. Third, responsibilities of software developers have reduced thanks to advancements in developer tools and provided frameworks. For example, Java platform, Enterprise Edition (Java EE) provides dozens of Application Programming Interfaces (APIs) including security features, data transfer (e.g. WebSocket), persistence, client-server connection frameworks such as JavaServer Faces (JSF) and more.

The research described herein aims to utilize the latest developments in cloud software engineering in the area of visual M&S. This thesis presents a visual M&S Integrated Development Environment (IDE) built on top of the Java EE-based client-server architecture. Traffic networks area is selected as the example problem domain to illustrate the capabilities of our IDE. Our research project is named as **C**loud-based visu**A**l simulatio**N** en**V**ironment for tr**A**ffic network**S** (CANVAS), which can be accessed at http://orca.cs.vt.edu/canvas.

1.2 Statement of the Problem

Traditionally, M&S applications have been developed under the SaaP paradigm. Advancements in cloud computing have triggered the shift towards SaaS in M&S research area. Recently, simulation software developers have been moving towards the development on the cloud. Yet, the need for Integrated Development Environments (IDEs) to develop visual simulations under the SaaS paradigm has not been fully satisfied. Visual simulation IDEs created under the SaaS paradigm provide easy access to the software, less hardware dependency, and easier collaboration with other developers. Creating a cloud-based simulation environment with up-to-date cloud technologies can address the lack of visual M&S tools under the SaaS paradigm and contribute to a paradigm change in M&S research area.

1.3 Statement of the Objectives

The research described herein aims to accomplish the following objectives:

1. Create an IDE for visual M&S of traffic networks, an example problem domain, as a cloud software application under the Java EE-based client-server architecture.
2. Enable the use of the IDE under a web browser such as Chrome, Safari, or Firefox on any network-connected laptop or desktop computer.
3. Develop a Conceptual Framework to guide the modeler for the construction and execution of a cloud-based visual M&S application in the traffic networks problem domain.
4. Develop an asynchronous visualization and communication strategy under the client-server architecture to provide adequate performance.

1.4 Overview of the Thesis

Previous research on cloud-based visual simulation tools as well as visualization technologies and available data storage/exchange formats are discussed in Chapter 2. Chapter 3 presents the architecture of CANVAS with its components and the responsibility of each component. Cloud-based simulation visualization can be conducted with different strategies. Chapter 4 presents categories of cloud-based simulation visualization and our strategy for simulation visualization. Being a traffic network simulation tool, CANVAS must provide a logic for the realistic representation of a traffic network. Chapter 5 presents how CANVAS simulates a traffic network along with the details of its conceptual framework and graph-based algorithm. Chapter 6 explains the implementation of CANVAS components in detail. Even though CANVAS is a simulation environment for non-experts, it is still useful to give guidance for the proper design of a traffic model. Chapter 7 gives example case studies to educate users. Chapter 8 presents the conclusions and future research.

1.5 Summary of Contributions

The research contributions can be summarized as follows:

1. Created an IDE for visual M&S of traffic networks as a cloud software application under the Java EE-based client-server architecture.
2. Enabled the use of the IDE under a web browser such as Chrome, Safari, or Firefox on any network-connected laptop or desktop computer.
3. Developed a Conceptual Framework to guide the modeler for the construction and execution of a cloud-based visual M&S application in the traffic networks problem domain.
4. Developed an asynchronous visualization and communication strategy under the client-server architecture to provide adequate performance.
5. Implemented a user account system enabling users to sign in and store their projects under their accounts. Users also can make their projects public for other users to use and modify.
6. Developed a model storage and retrieval strategy considering screen resolutions to enable visualization on any client desktop or laptop computer.
7. Implemented a file upload feature to enable users to upload an aerial photo of a traffic network to use as a background image in visual model construction.

Chapter 2: Literature Review

2.1 Cloud-Based Visual Simulations

"World Wide Web technology has the potential to significantly alter the ways in which simulation models are represented, developed and executed." [Kim *et al.* 2002 p. 230] Researchers has focused on implementing simulation languages, providing simulation services, and utilizing existing technologies in the area of web-based simulation since the early times of the web. This literature review investigates research conducted on industry-standard technologies that are utilized to implement cloud-based M&S tools. Note that "web-based" and "cloud-based" terminologies are used interchangeably. "Cloud-based" is more commonly used in the recent years.

Lorenz *et al.* [1997] present one of the early solutions in web-based simulations. HTML forms are used to collect input from users in their solution. An HTML page is transferred back to the client after it is generated by CGI-Scripts using the simulation result. The data is transferred through a Common Gateway Interface. A web-based simulation center to support communication among simulation projects is designed by Henriksen *et al.* [2002]. They use PHP enabled web-server and MySQL on the server side. HTML and JavaScript are used on the client side enabling multi-platform and multi-browser support.

One of the major technologies used for virtualization purposes had been Java Applets since it was able to provide app-like features within web browsers. Salisbury *et al.* [1997] present a web-based simulation visualization architecture with two main components: Simulation Monitor and Simulation Viewer. Former is a server-side HLA-compliant module that receives data from multiple objects. Collected data is sent to clients to be displayed in Simulation Viewer, which utilizes Java Applets and Java3D API. De Lara and Alfonseca [2001] implement the object oriented continuous system modeling program (OOCSMP) that creates Java Applets from simulation models to display active presentations in web-documents. Gan *et al.* [2001] developed a parallel supply chain simulation in which a PERL program initiates a simulator in the server and records event traces to multiple files. These files are sent to the client through a TCP/IP connection, and results are displayed in Java Applets. Graupner *et al.* [2002] present an architecture for simulation of manufacturing systems through the internet. The architecture fully decouples the simulation and data domains. Connections between domains are provided by Java-RMI. Java Applets are used for the visualizations. Wang and Liao [2003] implement a collaborative web-based simulation and modeling environment in which an XML-formatted simulation model is generated by the client. The model is converted into DEVS (Discrete Event System Specification) based simulation code on the server side. The client displays the simulation results through Java Applets.

X3D, an ISO Standard declarative language, has played a significant role in representing 3D computer graphics. Researchers have used X3D in their tools and frameworks for virtualization purposes. Kim *et al.* [2002] create an XML-based M&S framework named Rube. Simulation capabilities are provided by SimpackJ/S [Park 2002], which is a toolkit for discrete event simulation. An executable 3D model is created by exporting the scene into a VRML file and translating it into an X3D file. Boukerche *et al.* [2008] present the LAViE-3D framework for web-based distributed simulation visualization. They use X3D to display real-time 3D graphics,

and AJAX technology to interact with the X3D scene. Network communication is provided by XMLHttpRequest API. The federate application, which handles the data coming from HLA federation and user interactions, is written in Java.

There have been many other technologies utilized to develop web-based simulation tools. Jeong *et al.* [2009] present a web-based simulator for supply chain management using JSP and MS-SQL. Simulation results are saved to database and users can display the outcome through a web-browser. Gocmenoglu and Acarman [2014] present a 3D web-based visualization tool that reads JSON formatted data and displays the simulation on a WebGL canvas. Padilla *et al.* [2014] present ClouDES, a discrete-event simulator with the purpose of making the simulations accessible by non-experts. They utilize the existing simulation engine Desmo-J, and the Meemoo Graph Editor to facilitate the graphical design of the model.

Remote simulation/local visualization is one of the subcategories of web-based simulation that combines benefits of running simulations on a powerful server computer and visualizing the results on a client device. Myers and Balci [2009] present a web-based visual simulation architecture that follows the remote simulation/local visualization paradigm. They use Java for backend operations. XML formatted data is sent as part of HTTP requests and Scalable Vector Graphics is used for visualization in a web browser. Mwalongo *et al.* [2015] develop an application that remotely visualizes data from molecular dynamics simulations. They use WebSockets for client-server communication and WebGL for visualization on the client machine.

2.2 Cloud Software Development Platforms

Developing an end-to-end cloud-based simulation environment requires implementation in all tiers of the client-server architecture: Client, Web, Business, Data-Mapping, and Data Source. For this reason, a wise choice of a development platform is important since it can greatly reduce the workload of a software engineer. Today, there are many popular platforms that offer easier development for software engineers. Java EE, Microsoft .NET, Ruby-on-Rails, and Node.js are only a few of them. A comparison of these platforms is out of scope of the research presented herein. A comparative overview of cloud software development platforms is provided by Schutt and Balci [2016].

We have developed CANVAS using Java EE which offers extensive development technologies. JavaServer Faces, Persistence API, WebSocket API, and Enterprise JavaBeans (EJB) are only a few of the technologies that we have used in CANVAS. Cross-platform operability of Java EE, and the powerful capabilities of Java for backend operations are contributing factors to our selection.

2.3 Visualization Technologies

Improvements to visualization capabilities of web browsers have fostered the shift towards cloud-based simulations from stand-alone simulation applications. There are many advanced tools to serve as a visualizer within web browsers, but some of them are not considered in our

architecture. A visualization tool would have to have certain properties to be used in our architecture. First, the tool has to be platform independent. For example, Microsoft DirectX provides rich multimedia elements such as full-color graphics, video, 3D animation, and audio, but it is specifically designed for Windows-based computers [Microsoft 2018]. Second, plugin-free technologies are preferable since plugins increase the dependency on their proprietors. As an example, the Adobe Flash Player plugin is one of the most popular tools that delivers browser-based contents and applications [Adobe 2018]. However, we do not consider utilizing Adobe Flash Player to implement CANVAS as a plugin-free tool.

"Any modern web-based system should take the available standards put forth by the W3C into consideration, and a web-based simulation is no different." [Myers 2004 p. 38] Following web standards makes applications available to wide variety of devices while reducing the development cost. Therefore, the preferred method is to be able to provide simulation visualizations without requiring any specialized software. As a result, royalty-free web standards are the best options to provide visualization capabilities within the cloud-based simulation architecture. The WebGL standard as well as X3DOM and Three.js libraries that are higher level than WebGL are considered the main graphics providers of the architecture.

2.3.1 WebGL

"Web3D consortium long time ago introduced technologies like VRML and X3D, but new HTML 5 standard is opening new possibilities for visualizing 3D objects. Build-in WebGL and Canvas components controlled by JavaScript code running inside the browser window can vanish last differences between desktop and web applications." [Sawicki and Chaber 2013 p. 662] WebGL is a 3D graphics API based on OpenGL ES specification and supported by all major browser vendors. It brings powerful visualization capabilities of desktop applications to web browsers.

WebGL uses the HTML canvas element. The canvas element creates a specific area within web pages that can be used to build animations or visualizations. Developers can set the specifications of a canvas element such as its size and its styling using HTML and CSS. JavaScript is used as the development language to create 2D or 3D graphics within the canvas.

While WebGL is a powerful tool, developing a program from scratch with WebGL is time consuming. JavaScript frameworks such as X3DOM [2018], BabylonJS [2018] and Three.js [2018] are developed to abstract the details of the implementation. We prefer using higher level libraries to save time spent implementing visualization engines.

2.3.2 X3DOM

X3D is an open standards file format that provides 3D scene creation within web browsers. It is the successor of Virtual Reality Modeling Language (VRML) and developed by the Web3D Consortium. Its supported file formats are XML, Compressed Binary, and VRML [Web3D 2018]. X3DOM provides 3D content and makes X3D elements part of HTML DOM. The main difference between X3DOM and other JavaScript libraries is that X3DOM provides content in a

declarative way, while other JavaScript libraries are imperative. A scene that has a box geometry in it can be defined as in Figure 1.

```
<x3d width='500px' height='400px'>
    <scene>
        <shape>
            <appearance>
                <material diffuseColor='1 0 0'></material>
            </appearance>
            <box></box>
        </shape>
    </scene>
</x3d>
```

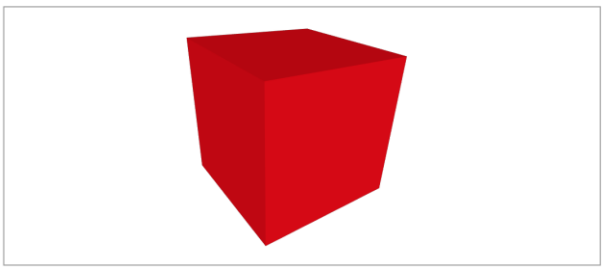

Figure 1: Box Geometry and its Implementation in X3DOM [X3DOM 2018]

X3DOM contains a few disadvantages comparing to libraries such as Three.js. First, the scene can be extremely wordy if there are many objects. Second, it is easier to maintain the scene with imperative programming, because logic can be installed into an object and those objects can make decisions according to the current state of the simulation. For example, vehicles in CANVAS can stop when they get close to a traffic light just by checking the distance. Even though X3DOM provides a good use of WebGL, its declarative nature was a disadvantage for a traffic simulation visualization system.

2.3.3 Three.js

Three.js and BabylonJS are both JavaScript libraries built on WebGL, and they provide similar visualization capabilities. While we do not have any performance benchmark, we selected Three.js because of its higher popularity and vast number of example projects [Three.js 2018].

A visual simulation can have hundreds of objects that have different specifications. Three.js is an efficient framework to set the details of simulation objects. Since a Three.js mesh is a JavaScript object, custom logic can be added into the mesh without requiring special programming knowledge for the library. A box geometry can be added into a Three.js scene as follows:

```
var geometry = new THREE.BoxGeometry( 1, 1, 1 );
var material = new THREE.MeshBasicMaterial( {color: 0x00ff00} );
var cube = new THREE.Mesh( geometry, material );
scene.add( cube );
```

Figure 2: Adding Box Geometry into a Three.js Scene [Three.js 2018]

We have chosen to use Three.js as the visualization technology of CANVAS for multiple reasons. First, it abstracts low-level details that would need to be implemented if WebGL was used. This lets us focus more on application logic instead of trying to implement details of graphics in web browsers. Second, it makes our simulation engine extremely compatible with

our visualization engine. The simulation engine programmed by Java runs on a server computer and sends the result to the client device for visualization. Events and objects wrapped by the simulation engine can be unwrapped and visualized by the visualization engine in an efficient manner thanks to Three.js.

2.4 Model Storage and Data Exchange Formats

Text-based data exchange formats are extremely useful to share data between platforms. Many of the programming languages support these formats and provides serialization and deserialization libraries. Two of the widely used formats are Extensible Markup Language (XML) and JavaScript Object Notation (JSON). They both have advantages and disadvantages over each other, and we have considered both technologies find the best fit for CANVAS.

A CANVAS simulation instance continuously sends data from server to client and there may be a lot of simulation instances running at the same time. Therefore, using JSON provides faster load and less bandwidth usage since it is less wordy comparing to XML format. CANVAS vehicle array representations are provided as examples in XML and JSON formats for comparison purposes in Figure 3 and Figure 4.

```
<vehicles>
    <vehicle>
        <id>d1</id> <speed>40</speed> <length>12</length>
    </vehicle>
    <vehicle>
        <id>d2</id> <speed>50</speed> <length>11</length>
    </vehicle>
    <vehicle>
        <id>d3</id> <speed>45</speed> <length>13</length>
    </vehicle>
</vehicles>
```

Figure 3: Vehicle Array Representation in XML

```
{"vehicles":[
    { "id":"d1", "speed":"40", "length":"12" },
    { "id":"d2", "speed":"50", "length":"11" },
    { "id":"d3", "speed":"45", "length":"13" }
]}
```

Figure 4: Vehicle Array Representation in JSON

XML can be integrated with other markup languages such as HTML easily. XML has a tree structure as HTML while JSON has a map structure. If we had used X3D file format to represent scenes, XML could be useful to modify and manipulate HTML DOM tree. Since we use JavaScript extensively in the visualization engine, JSON is more applicable in the CANVAS architecture.

Cloud-based visual simulation tools must have a data exchange mechanism between client and server. In CANVAS, JSON format is not only used to store models but also to share data between client and server continuously. Since JavaScript can serialize and deserialize JSON efficiently, a data exchange mechanism based on JSON format is the best option for CANVAS.

Chapter 3: CANVAS Design

In CANVAS, we define "environment" as a set of integrated software tools that provide computer-aided assistance throughout the development and execution of a cloud-based traffic network simulation model. We have developed CANVAS under the Java EE-based client-server architecture, assigning responsibilities to each of the integrated tools. The account system, project creation and management, and user collaboration are some of the provided functionalities that aid development of traffic simulation models. In this section, we present the design of CANVAS with its hardware and software development environment, architecture, tools, and client-server communication strategy.

3.1 Hardware and Software Development Environment

CANVAS is a Java-based cloud software application that is built using the Java EE platform and its specifications. Several Java EE APIs including Enterprise JavaBeans, JavaServer Faces, Java Persistence, WebSocket, and JSON Processing are used to develop CANVAS. CANVAS is deployed on a server computer to run under the GlassFish application server. MySQL relational database management system is used to store and retrieve user information and project data. The server is a PowerEdge T330 server computer with 64GB RAM and 480GB solid state hard drive running the CentOS Linux operating system.

3.2 Architecture

"An architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution." [IEEE STD 1471-2000] The CANVAS architecture is developed on top of the Java EE client-server architecture which consists of five tiers: the client tier, web tier, business tier, data mapping tier, and data source tier. Specifications provided by Java EE can handle most of the heavy work of a large cloud software application enabling software engineers to focus on the application logic. Figure 5 presents an architectural overview of CANVAS. The responsibilities of each tier are discussed in the following sections.

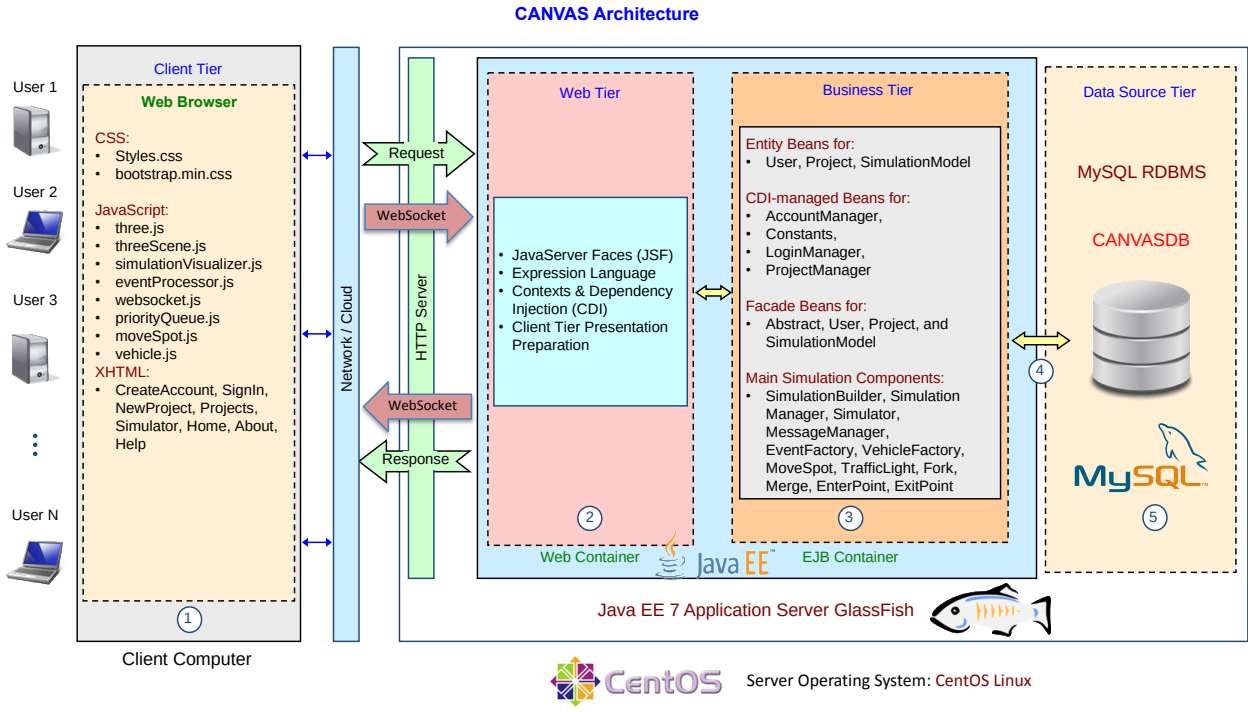

Figure 5: CANVAS Architecture

3.2.1 Tier 1: Client

The client tier consists of application clients that access a Java EE server and that are usually located on a different machine from the server [Oracle 2018]. Usually, a client can be a browser, desktop application or another server computer. CANVAS is designed to serve clients over a web browser.

The client tier provides the interaction between clients and the application. The presentation and user interface are mainly provided by XHTML and CSS code with some JavaScript code to change the display according to user interactions. The client logic is developed in JavaScript utilizing the Three.js library for the development within an HTML canvas element. Three.js provides functionality for model composition, simulation visualization, and user interactions on the HTML canvas element.

3.2.2 Tier 2: Web

The web tier does the heavy work through JavaServer Faces and Context & Dependency Injection (CDI). Enterprise JavaBeans can be used effectively within Java EE web applications thanks to services provided by CDI. Developers do not need to create entity objects and deal with object factories when CDI is used. For example, a session bean is initialized at runtime, and an EJB container stores the object reference. This object can be used within an enterprise java bean without an explicit initialization.

Expression Language (EL) provides a communication mechanism between the web tier and enterprise java beans. EL can be defined within XHTML tags to present data provided by business tier. EL can be used for multiple purposes such as displaying a value from the backing bean. It can also simply call a Java method with a button click on the user interface. The client-server connection and request/response mechanism are provided under the hood by JSF.

JSF provides all client operations except simulation runtime operations in CANVAS. Full duplex WebSockets (WS) are used for data exchange during simulation runtime. Figure 5 indicates that client-server communication is provided both by HTTP Request/Response and WebSockets. JSF is used for operations such as account creation, user sign in, project and simulation model storage, project retrieval.

3.2.3 Tier 3: Business

The business tier provides the core functionality of a Java EE-based cloud software application. The Java EE specification offers several technologies in this tier some of which are used in CANVAS. EJB and the Java Persistence API are used to create the base of business logic in CANVAS. EJB is used to provide dynamic data to the web tier as explained in Section 3.2.2. The Java Persistence API handles object storage and retrieval in the business tier. Structured Query Language (SQL) queries can be embedded in Java code to fetch desired data into the business tier from the database. The entity manager interface provides a robust mechanism to find, store, remove, or edit data in the database through the business tier. All business logic except simulation runtime operations is provided through EJB and Java Persistence API.

The rest of the business logic is implemented in plain Java code. It contains static and dynamic traffic network simulation objects such as vehicles, move spots, traffic lights, forks, and merges. Simulation components such as simulation builder, simulation manager, and message manager are also part of the business tier. Unlike EJB components, simulation data communication is provided by bi-directional WebSockets for simulation execution.

3.2.4 Tier 4: Data Mapping

Java EE offers several APIs for data mapping including the Java Persistence API and Java Database Connectivity (JDBC). JDBC is the backbone of the connection between the Java programming language and SQL databases. The connectivity must be active to run the application server. Developers can execute SQL queries on JDBC to create database tables. The Java Persistence API provides a mechanism to create entity classes by reading the tables in relational database management system (RDBMS). CANVAS stores information about Users, Projects, and Simulation Models. Their entity classes are created by the Persistence API. Moreover, session beans, which are used to run queries on the entity manager and return data from the database, are created through the Persistence API. Overall, Java EE provides extremely efficient services for engineers to handle data mapping.

3.2.5 Tier 5: Data Source

Java EE supports many relational (e.g. MySQL) or non-relational (e.g. MongoDB) databases. CANVAS uses a relational database management system for a couple of reasons. First, CANVAS always stores consistent and structured data for users and projects. Therefore, the data flexibility of NoSQL databases is not an advantage for this system. Second, the horizontal data scalability of a non-relational database is not required in CANVAS since a single powerful server computer provides the necessary computer resources. SQL queries facilitate safe and efficient data manipulation in CANVAS. Therefore, we have chosen to use MySQL, a RDBMS, among many other options such as Oracle, Java DB, PostgreSQL.

CANVAS has a file storage system in addition to the RDBMS. The file storage provides storage for aerial photos or maps of areas that simulation models are built on. Unique file names are generated in the business tier before files are saved into the storage system. File names are stored as part of projects in RDBMS.

3.3 CANVAS Components

CANVAS's component-based architecture assigns a single responsibility to each component. This approach provides high cohesion within components and low coupling between components. The model composer and visualization manager run on the client-side, while the simulation builder, simulation manager, and message manager run on the server-side. JSF and JavaScript are used to serve user interfaces and serve to the client for actions such as creating a project, building a simulation model, and saving and retrieving a composed model. WebSockets serves as intermediary communication technology between the client and server during simulation runtime. Message buffers, placed in both client and server devices, are in interaction with each other to follow the data flow and pause or stop the simulation when necessary.

The simulation visualization process starts by composing the model. A model has to be validated to be able to produce a valid executable input for the simulation manager. The simulation manager receives the built model as an input to execute the simulation. Data is sent to the client to be visualized through the message manager during simulation execution. The visualization manager interprets the data to present it to users.

The components of the architecture are discussed in the order of event flow in the following subsections. The simulation execution process completely bypasses the web tier. The client and server are connected through WebSockets during simulation execution. In this way, communication is provided over TCP/IP which enables clients to connect to the business tier directly. Figure 6 describes the relationships among the architecture tiers and CANVAS components within tiers. The responsibility of each component in is briefly described in the following subsections while implementation details are presented in Chapter 6.

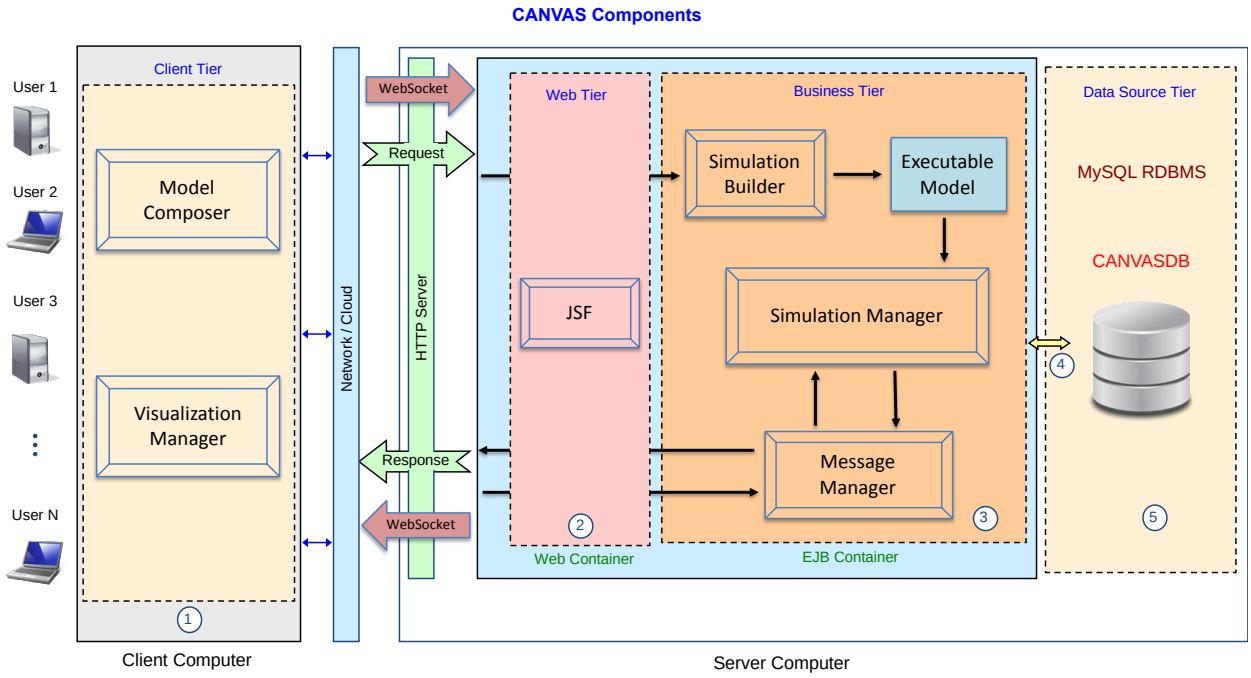

Figure 6: CANVAS Components

3.3.1 Model Composer

Simulation applications must have functionality for users to provide necessary inputs or files. CANVAS is a traffic network simulation environment that requires an aerial photo or map of the area to build a traffic network on. Therefore, we provide file upload functionality for users to upload their background maps before composing the simulation model. After initial project features are set, users can start composing their models.

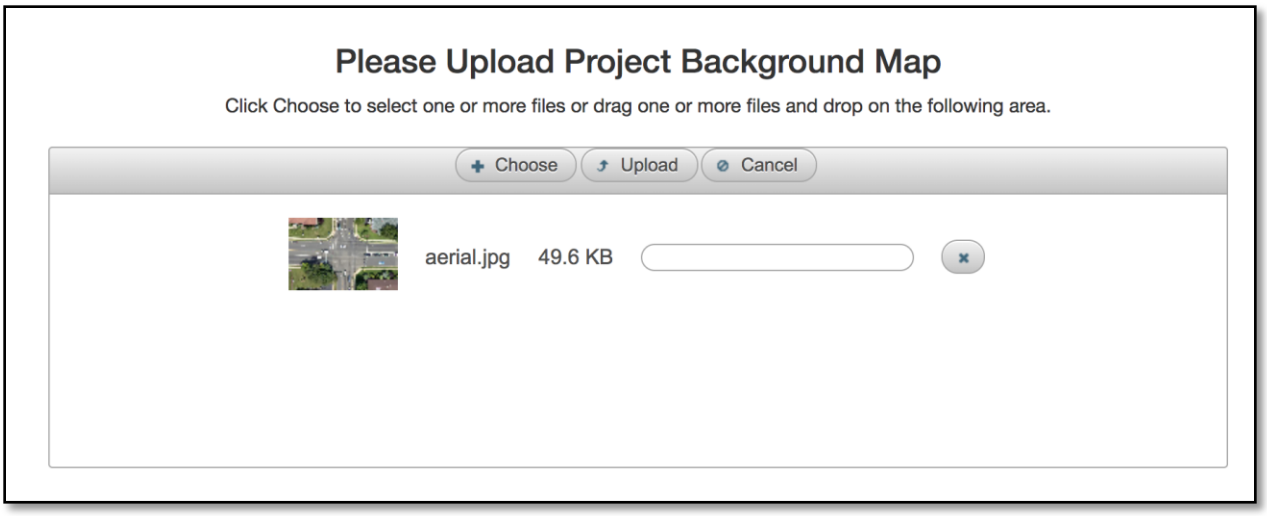

Figure 7: Background Map Upload Service

Model composers in end-to-end M&S tools must have well-tested, ready to use model components. The CANVAS Model Composer consists of the tools listed in Figure 8. A selected component is added to a clicked location within the simulation model.

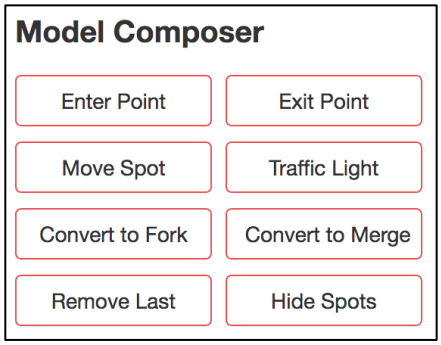

Figure 8: CANVAS Model Composer

CANVAS aims to simulate traffic networks as realistically as possible and offers flexibility for users to compose their simulation models. For this reason, some components of the simulation have to be modifiable. CANVAS updates the model composer interface dynamically when a component is selected. For example, users can set the timing of a traffic light before adding it to the simulation model as depicted in Figure 9.

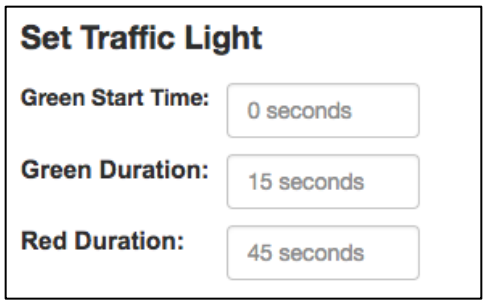

Figure 9: Traffic Light Settings in Model Composer

3.3.2 Simulation Builder

Model validation plays a vital role in M&S tools. If a simulation system is designed for non-experts, the chances of composing an invalid model can be considerably high. Accepting an invalid model might have fatal results. The execution might crash or continue running with unexpected results, which would damage the reliability of the simulation tool. Therefore, a model validator must provide a feedback to the client when simulation cannot be executed due to an invalid model specification. The CANVAS simulation builder validates the model while preparing it for the execution.

The model validator is imported inside the simulation builder in CANVAS architecture since a model can be built while validating it by parsing the input only once. The simulation builder receives the model composed by the user as input and parses it. It creates static simulation objects and connects them while security checks for model validation are conducted at the same time. This operation produces a graph representing a traffic network. The model validator has to be sure that a valid graph with valid objects are created and checks if road paths and intersections are created correctly. For instance, if there is a fork point on the model, it must have two connected move spots for vehicles to move. If the fork object is created incorrectly, the server informs the client about the invalid object.

3.3.3 Simulation Manager

The simulation manager is the core of CANVAS architecture. The traffic network simulation algorithms run inside this component with the input received from the simulation builder. It continuously produces visualization results and delivers them to the message manager. All events that can be visualized are produced by this component.

A thread-safe simulation manager instance is created for each client. Each simulation manager enters a waiting stage while the required number of events are generated. Therefore, system resources can be used for other clients. When a client consumes events, execution continues to feed the client with more events. In case clients pause the visualization, the simulation manager automatically enters a waiting stage since the client stops requesting new messages.

3.3.4 Message Manager

The message manager has a buffer that stores messages produced by the simulation manager before sending them to the client. JSON formatted messages are stored in the buffer until the client requests them. Since the client is in control of the data flow, it cannot be overwhelmed by messages sent from the server. More information on data flow is provided in Chapter 4.

The messages must be in order of time before sending them to the client. However, the simulation manager does not necessarily produce messages in real time order. For example, at current time *t*, an enter point can schedule a vehicle creation event for time *t+10*. At time *t+2*, a vehicle direction change event might be produced by another object for time *t+3*. In this case, the direction change event must be sent to the client before the vehicle creation event even though it is generated later.

The ordering of the messages is implemented in the message manager. A buffer that uses a priority queue data structure is utilized for message ordering.

3.3.5 Visualization Manager

The visualization manager visualizes messages produced by the simulation manager. Since static objects do not move, the main responsibility of the visualization manager is to update coordinates of dynamic objects. Each dynamic object has a speed and rotation which is used to move the object in every time frame.

The visualization manager has a buffer that keeps the events in order. When the earliest event time gets equal to the current visualization time, the event is processed. If there is not a new event in a certain time frame, the visualization manager only updates the coordinates of dynamic objects according to their speed and direction. The visualization manager can process different type of events. For example, if it is a vehicle creation event, a vehicle at the specified coordinates is created. If it is a speed change event, the speed of a dynamic object is updated.

The visualization manager also has other responsibilities such as assigning the image of the object. CANVAS provides many vehicle images. If newly created object is an automobile, an automobile image is assigned to the object. If it is a long vehicle, a truck or bus image is assigned. The visualization manager also changes the state of traffic lights as green or red.

3.4 Client-Server Communication Strategy

Component-based design of a cloud-based simulation provides multi-platform support, easier maintainability and replaceable components when newer technologies arise. A software developer can inherit an architecture partially and use different components for other parts of the architecture. To illustrate, CANVAS is built on Java EE, and the simulation engine is written in Java. A simulation engine running on C# can be replaced with the current simulation engine since other components of the architecture are fully decoupled. This approach improves the code reusability and systems components can be replaced without making modifications on the other

components. The client-server communication strategy of CANVAS is developed considering these concerns.

The client-side implementation of CANVAS includes a model composer and visualization manager while the server-side has a simulation builder and simulation manager. The communication between the client-side and server-side components are provided by implementing an asynchronous visualization protocol. The protocol utilizes WebSockets that enable continuous connection during the session.

WebSockets are modified HTTP-compatible sockets designed specifically for the web. While sockets can be used for any client application, WebSockets only work within web browsers. After HTTP-handshake is made between the client and server, WebSockets provide continuous connection until one of the parties terminates the connection. This has a significant advantage over HTTP, since HTTP requires a handshake every time data is sent. Moreover, WebSockets use less network resources since they eliminate the need for extra messaging between the client and server.

Oracle published the WebSocket API with Java EE 7 that abstracts the details of WebSockets implementation. Therefore, a fast and secure implementation with WebSockets is possible in Java EE platform. A code snippet is provided in Figure 10 to demonstrate a WebSocket connection in Java EE. The client-side must provide a URL to start a connection with the server computer.

```
Client:

var socket = new WebSocket("ws://shark.cs.vt.edu/TrafficSimulator/actions");

Server:

@ApplicationScoped
@ServerEndpoint("/actions")
public class WebSocketServer {
       .
       .
       .
}
```

Figure 10: WebSocket Client-Server Connection

WebSockets are used to implement the communication of CANVAS transferring JSON formatted data between the client and server.

Chapter 4: Asynchronous Visualization Protocol

Researchers have been proposing different approaches to tackle the challenges in simulation visualization on cloud-based platforms. Enabling user interactions, reducing network latency, managing server resources, and providing availability for various client hardware are only a few of the problems. There are different protocols and strategies that address particular problem well but proposing a system that solves all the challenges is not an easy task for researchers.

Some Cloud-Based Simulation (CBS) tools have very limited user interaction capabilities. Many simulation tools get the simulation model as input and process it at the server and return results to the client. This approach makes user interruption impossible. In this case, the simulation execution has to start over if the user wants to make a change on the model. Some solutions which enable user interactions suffer from the network latency. Users have to wait for the simulator to process the execution in addition to the time spent on data delivery. Even a few seconds of pause on the simulation visualization bothers the today's average internet user. A good solution to address this problem would be running the simulation execution on the client device, but this would consume an unaccepted amount of resources on the client device. Moreover, most of the benefits of the cloud-based simulation visualization cannot be utilized with this approach.

Efficient management of server resources plays a significant role to be able to serve clients concurrently. A CBS system must be scalable and wisely use its resources. Many of the simulations use high amount of resources which makes the resource management even more vital. Some simulation tools provide dedicated resources for users enabling active users to interact with the simulation environment with minimum latency. However, new users have to wait until some of the active users leave the simulation environment when there are many concurrent connections. A better approach for cloud-based visual simulation tools would be pausing the simulation execution after it stores enough data to serve the client. In this way, new users can directly start using the server resources. The paused thread can continue after the user consumes some part of the data.

Researchers have categorized the simulation visualization approaches. In the following subsection, cloud-based simulation categories with their advantages and disadvantages are presented. In Section 4.2, the visualization strategy of CANVAS is described in detail.

4.1 Cloud-Based Simulation Categories

Three of the main categories describes how cloud-based simulation tools can be developed architecturally [Byrne *et al.* 2010]. These are remote simulation visualization, local simulation visualization, and remote simulation/local visualization. We briefly describe each of them to give a quick overview of the cloud-based simulation categories. Detailed explanations and evaluations can be found in the research conducted by Myers [2004] and Byrne *et al.* [2010].

4.1.1 Local Simulation Visualization

The local simulation visualization technique gives the responsibility of both simulation execution and visualization to the client machine. When users connect to the simulation environment's website, the simulation engine and visualization engine are downloaded to the client device. Researchers had utilized technologies such as Java Applets and Adobe Flash, but today all functionality can be developed by JavaScript and a visualization technology such as WebGL.

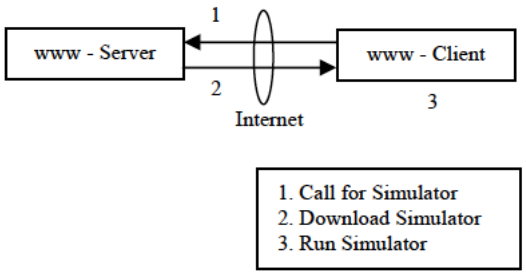

Figure 11: Local Simulation Visualization [Myers 2004]

Server computer responsibility can be highly reduced per user with this approach. Thus, maximum number of users can be served at the same time. Furthermore, it completely removes the network latency after engines are ready. However, this approach reduces the many advantages of CBS such as utilizing the powerful server computers and providing highly sophisticated simulation tools to client devices which have low computing capabilities.

The local simulation visualization does not offer more than an application that is installed to a client device except that it can be used within a web browser. This technique offers significant advantages, but it does not meet some of our requirements such as making the cloud-based simulations available for wide range of users.

4.1.2 Remote Simulation Visualization

The remote simulation visualization technique assigns all responsibility to the server computer. In this approach, users have to wait until simulation visualization is finished and sent it back to the client device. Users basically display the results generated and visualized by the server computer. In this approach, the client device does not have a responsibility other than providing an interface for the model composing and displaying the result. Therefore, users can use the simulation tools built by this approach from devices which have low resources.

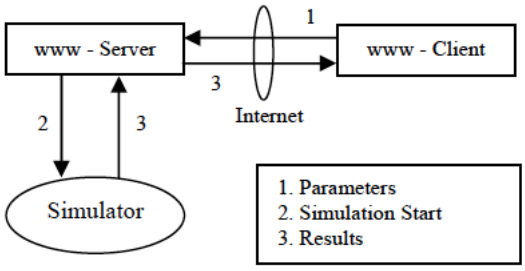

Figure 12: Remote Simulation Visualization [Myers 2004]

This technique assigns the heavy work completely to the server machine, which possesses significant disadvantages. First, the server machine can be loaded very quickly causing new users to wait. Second, users have to wait to get the visualization results after they make a visualization request. Third, this approach does not provide an efficient user interaction. In a locally visualized simulation, users can take an action any moment such as restarting the simulation or modifying the model. However, users have to wait for downloading the results to display them in this case, which is inefficient in terms of time.

4.1.3 Remote Simulation/Local Visualization

The remote simulation/local visualization is another technique that the simulation execution is conducted on the server while the visualization process is run by the client. Terms such as "hybrid simulation visualization" or "hybrid client/server simulation" are also used to define this category [Byrne *et al.* 2010].

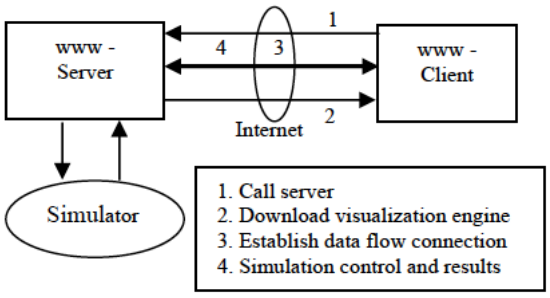

Figure 13: Remote Execution/Local Visualization [Myers 2004]

The hybrid technique requires more complicated design comparing to the previous categories. The responsibilities are shared between the client and server to make use of both devices at the same time. The simulation execution runs on the server device while the client runs the visualization. The client interprets the data sent from the server to run the visualization. All events and decisions are produced by the server and the state of the simulation is completely determined on the server device. This would reduce the resource usage on the client device significantly. The client-side visualization engine interprets the data sent by the server to generate visualization.

There are different protocols under the remote execution and local visualization category such as synchronous and asynchronous protocols. These protocols are compared in the next section and the asynchronous visualization strategy of CANVAS is presented in detail.

4.2 CANVAS Asynchronous Visualization Strategy

The visualization strategy of CANVAS is developed by using the remote simulation/local visualization technique. Therefore, a connection mechanism between the server and client must be provided to transfer data several times during a session. CANVAS utilizes the WebSockets API of Java EE, which provides efficient data transfer between the JavaScript on the client, and Java on the server.

Multiple actions are taken to present the visualization to the client in CANVAS. In the first step, a simulation model is sent to the server computer after it is composed on the client device. The simulation execution is completely processed on the server. For example, a traffic simulation model has multiple objects such as enter points, exit points, and traffic lights. The enter point generates a "vehicle creation" event or a traffic light generates a "change the traffic light state" event. The results of these events affect the state of the simulation and future events. Produced events are added to the buffer in the message manager with their time stamp before they are sent to the client. The earliest event is processed when the visualization time on the client gets equal to the event's timestamp. If it is a vehicle creation event, a vehicle image on the specified coordinates is created before rendering the page. Then, the user can view the vehicle image in the next time frame.

Multiple strategies can be developed to make use of the remote simulation/local visualization. In synchronous protocol, the server computer sends messages to clients according to its own state without knowing about the state of clients. Different algorithms can be used to manage the data flow in this protocol. For example, the message manager might send a chunk of data after it stores a certain number of messages. Another approach would be sending data in a certain time period such as sending a hundred messages in every two minutes. However, the client can be easily overflowed with the data since this approach completely ignores the state of the client.

CANVAS uses the asynchronous visualization strategy which gives the responsibility of data request to the client. A client has a smaller buffer that stores events to be processed. When the number of messages in the buffer gets smaller than the limit, the client makes a new request from the server. The message manager sends a chunk of messages from its own buffer. In this way, the client is in control of the data flow which reduces the memory usage. However, this approach also has a downside. Every time client makes a request, a new thread on the server is created. The number of messages should be arranged wisely so that the client does not request messages frequently. This approach keeps the client under the control to not get overflowed.

Users watch the simulation visualization in the real-time while the simulation manager can run the execution much faster than the visualization. For example, a server computer can execute hours of simulation in a few seconds. Therefore, the simulation manager should take extra precautions to not execute a simulation more than necessary. The user might only visualize a few

minutes of the simulation before leaving the simulation environment. Moreover, even though the client wants to visualize the whole simulation, finishing an execution much earlier would cause an extensive use of RAM since the data has to be stored a long time. Therefore, a logic should be implemented to pause the simulation execution when there is enough number of generated events. CANVAS is designed to pause the simulation execution after two thousand events are generated. The client requests one thousand events when the client buffer has less than a thousand events. When a data is sent to the client, simulation execution thread continues to run until the message manager buffer has two thousand events again. The buffer in the message manager is designed to store more events because of possible delays when the server has too many concurrent connections.

The CANVAS asynchronous visualization strategy provides many benefits. First of all, the server computer never uses unnecessary resources. It runs the execution as long as the client requests. Secondly, the user can pause and continue the simulation visualization without sending any request to the server computer. Since the visualizer stops requesting new messages when the user pauses the visualization, the thread on the server computer waits without consuming resources. With this approach, many users can use CANVAS at the same time.

A downside of the strategy is that an "event request thread" has to be created to request data. This is a separate thread than the execution thread, but in most cases, the execution thread is already in the waiting stage when a request thread is created because a simulation execution can be processed much faster than a visualizer can consume it. Therefore, the server computer does not have more than one active thread per user in almost in any cases. The event request thread has to wait until a number of events are generated. If there are less than one thousand events at the request time, the message manager sends all the events and the simulation manager continues to the execution to fill the buffer.

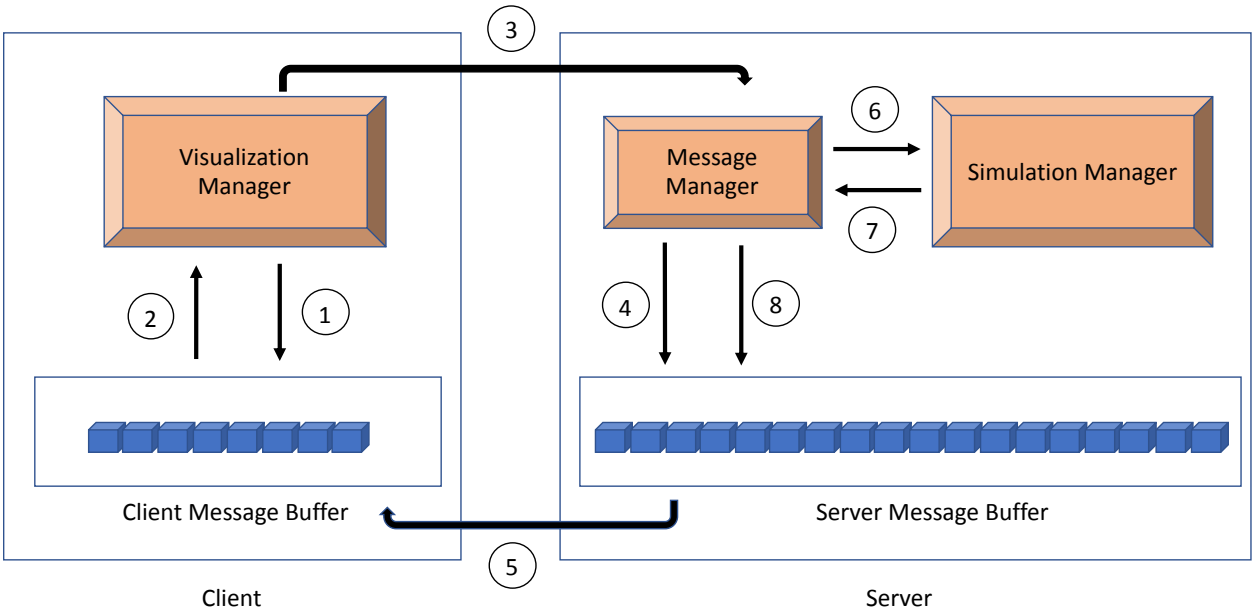

Figure 14: CANVAS Event Request

The event request from server is a complicated process. Note that a simulation instance is already created and it is running on another thread before an event request thread is generated. Therefore, this thread expects that there are already some messages in the server message buffer. If there is not, it waits until the message buffer receives some events from the simulation manager. The communication flow between the components when the visualization manager needs more events on the client-side message buffer is presented in the following.

1. The visualization manager asks for a new event from the front of the queue. It also checks the number of remaining events in the queue.
2. The client buffer returns the earliest event.
3. If the number of events in the buffer is less than the limit, the visualization manager invokes the message manager on the server.
4. The message manager checks if its buffer has event messages in it. If the buffer is empty it has to wait.
5. If there are less than one thousand messages, the buffer sends all events. Otherwise, it sends exactly one thousand messages to the client.
6. The message manager invokes the simulation execution thread since some of the results are consumed by the client.
7. The simulation manager returns the latest events to the message manager and continues sleeping until it is invoked again.
8. The message manager puts new messages into the buffer, which is a priority queue, by checking their time stamp.

Chapter 5: CANVAS Conceptual Framework

A traffic network simulation has to provide support for concurrent activities since several events happen at the same time. For example, a traffic light might change its state from red to green while a vehicle is turning left at the same moment. There are two possible ways to implement such a scenario. First, a thread can be assigned for each object in the simulation. However, this approach is unrealistic since a simulation might have hundreds of objects at the same time. Moreover, a cloud-based simulation environment must serve multiple users at the same time. The better approach would be to implement the logic in a sequential manner that can preserve concurrency of the events in the simulation.

The logic can be developed by facilitating a conceptual framework. "A Conceptual Framework (CF), also called world view, simulation strategy, or formalism, is a structure of concepts and views under which the simulationist is guided for the development of a simulation model. It is vitally important that a programmer follows a CF in implementing a simulation model in a HLPL (high level programming language)." [Balci 1988 p. 1] There are several conceptual frameworks such as "Event Scheduling", "Activity Scanning", "Three-Phase Approach", and "Process Interaction". The CANVAS conceptual framework is developed by implementing process interaction. More information about other conceptual frameworks can be found in research conducted by Balci [1988].

The process interaction conceptual framework consists of 4 phases: (1) initializations, (2) clock update, (3) scan, and (4) output. In the initialization phase, future events are generated with their time stamp. When the simulation time becomes equal to a generated event's timestamp, the scheduled event is processed. The clock update phase increments the simulation time. The scan phase is the phase where scheduled events are processed. Moreover, an event might be generated for the next time frame as a result of the current state of the simulation. To illustrate this for CANVAS, a vehicle might arrive at a fork spot and decides to turn left. The direction of the vehicle is changed, so that vehicle can move towards the new direction in the next time frame. The last phase is output. In CANVAS, users define a simulation duration time in terms of minutes. When a simulation execution exceeds the duration, enter points stop producing vehicles. CANVAS continues execution until all vehicles leave the simulation. At this point, CANVAS sends data collection results to the client. Figure 15 represents process interaction conceptual framework.

In the following, the logic behind simulating a traffic network is discussed. Section 5.1 describes the features and implementation of each simulation object. Section 5.2 presents the events in the simulation. Section 5.3 explains interactions among objects which generates the events.

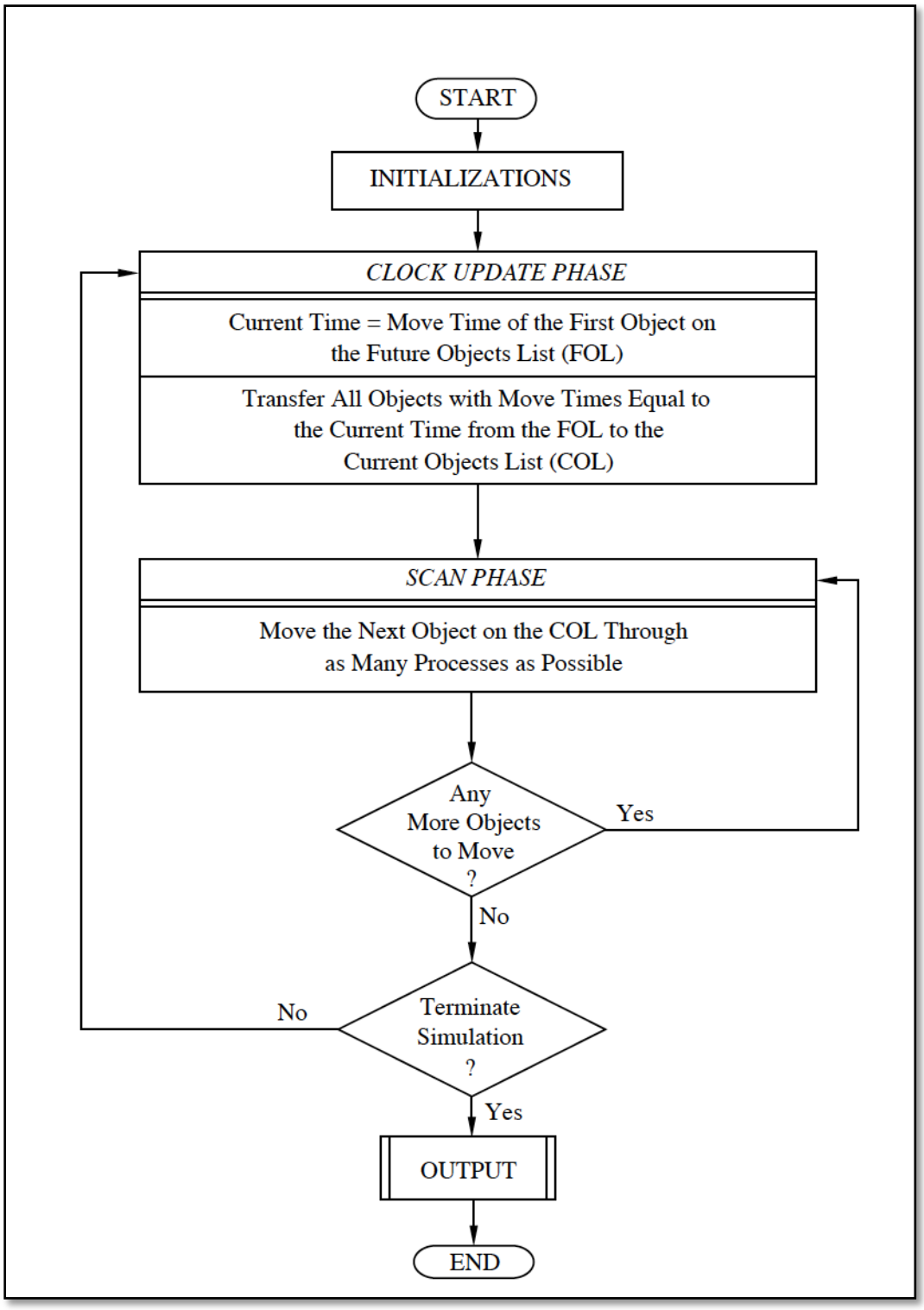

Figure 15: Process Interaction Conceptual Framework [Balci 1988]

5.1 Simulation Objects

Traffic network simulation objects can be classified under two main categories: static and dynamic. Static objects are the objects that define the traffic network structure (e.g. forks, merges, traffic lights) while dynamic objects are the objects that move within the structure according to their interactions with the other objects.

A model in CANVAS is a set of connected static objects that defines a graph data structure. The graph is created in the simulation builder before the simulation execution starts. The graph represents a network that dynamic objects can move on. In the following, simulation objects are classified and presented in detail.

5.1.1 Static Objects and Graph Representations

Static objects share common features while they also have some unique features that affect the behavior of dynamic objects differently. For example, if a traffic light is in the red state, vehicles arriving at this node have to wait until its state switches to the green state. A graph can have multiple end nodes which define the enter and exit points of the traffic simulation. The rest of the nodes are categorized as move spots, merges, forks, and traffic lights.

A traffic network model is always a **weighted directed graph** which can be either cyclic or non-cyclic.

Table 1. Graph Data Structure - Traffic Network Analogy

Graph Element	Definition	Traffic Network Analogy
Start Node	A node where there is not a directed edge pointing to it	Enter Points
End Node	A node where there is not a directed edge leaving from it	Exit Points
Inner Node	A node that has **at least** one incoming edge, and one outgoing edge	Move Spots, Forks, Merges, Traffic Lights
Directed Edge	The directed path between two nodes	Traffic lane where vehicles can move on a single direction
Weight of an Edge	A value assigned to an edge	The length of a traffic lane

Static objects in CANVAS are represented with different colors to define the type of an object. Table 2 presents the color to static object type mapping.

Table 2. Color to Static Object Type Mapping

Model Composer Color	Static Object Type
Orange	Enter Point
Black	Exit Point
Yellow	Move Spot
Blue	Fork
Purple	Merge
Green	Traffic Light in Green State
Red	Traffic Light in Red State

Figure 16: Basic CANVAS Model Screenshot

Figure 16 has an enter point, an exit point, and two move spots. Vehicles are produced in the enter point and they leave the simulation from the exit point. The graph representation of this model would be as in Figure 17. The weights of edges represent the distances between two points in the basic traffic network.

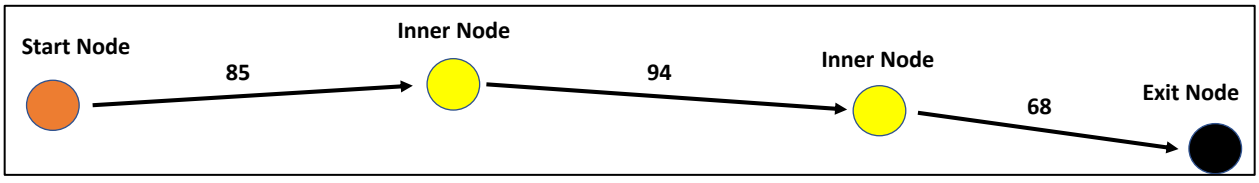

Figure 17: Basic CANVAS Model Graph Representation

The class hierarchy of CANVAS components is straightforward. Each object in the simulation has an id, x coordinate, and y coordinate. Therefore, the "StaticObject" and "DynamicObject" abstract classes extend another abstract class "SimulationObject". CANVAS makes use of polymorphism extensively since the connected objects do not know the type of the other objects. A neighbor of a "MoveSpot" can have any type such as "MoveSpot", "EnterPoint", "ExitPoint", "Merge", "Fork", and "TrafficLight".

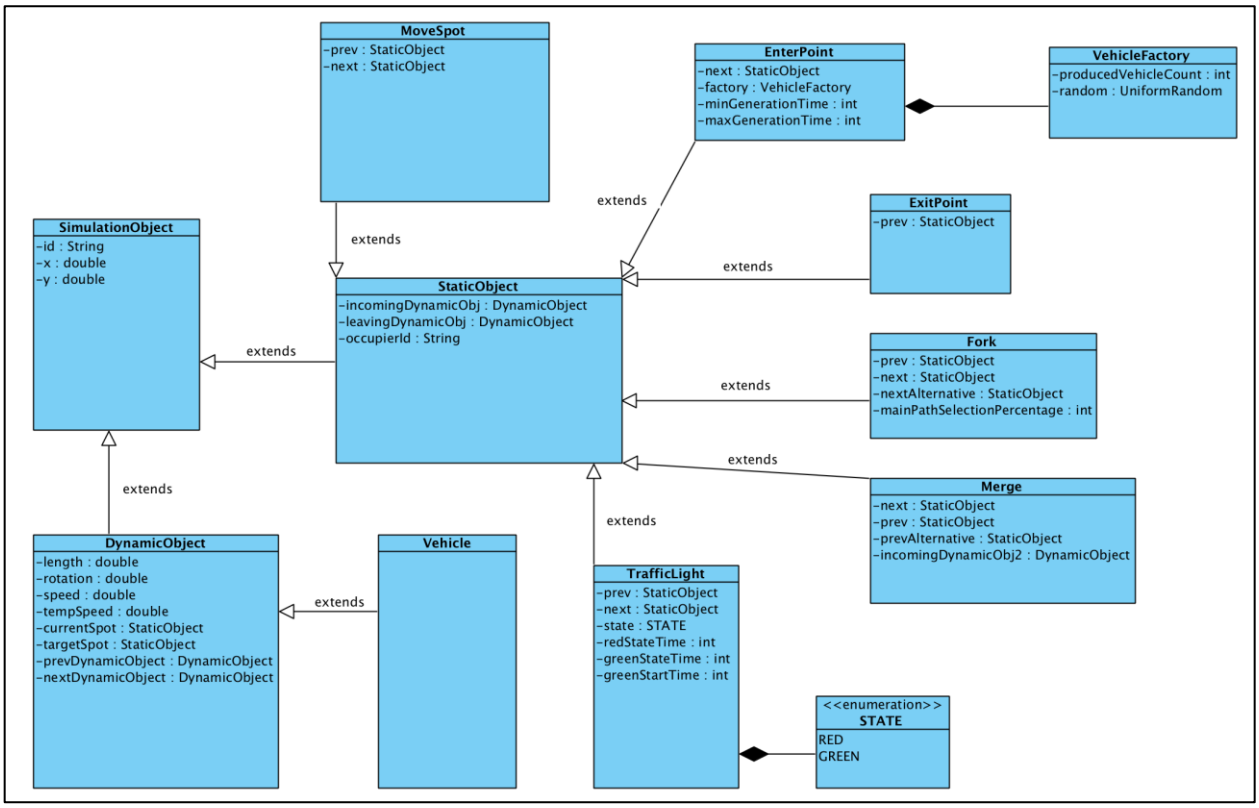

Figure 18: CANVAS Class Diagram for Simulation Objects

Static objects have the following attributes: "incomingDynamicObj", "leavingDynamicObj" and, "occupierId". The definition of each attribute is given in Table 3. The interactions between the static and dynamic objects are discussed in detail in Section 5.3.

Table 3. Static Object Attribute Definitions

Attribute	**Definition**
incomingDynamicObj	The closest vehicle moving towards to the static object
leavingDynamicObj	The vehicle that has left the static object most recently
occupierId	The id of the vehicle which occupies the spot. (If another vehicle gets close to the move spot, it has to wait until the occupier leaves)

5.1.1.1 MoveSpot

The move spot defines the paths of vehicles. When a vehicle arrives at a move spot, the connection between the previous static object and the vehicle is removed. The rotation of the vehicle is updated according to the new target spot of the vehicle, which is the "next" static object of the current spot. Users can add move spots to the simulation model as many as they prefer. If a road is curvy, the number of move spots should be increased, so that vehicles can change their direction more often that would ensure a more realistic simulation visualization. If a lane is straight the distance between the move spots can be much higher.

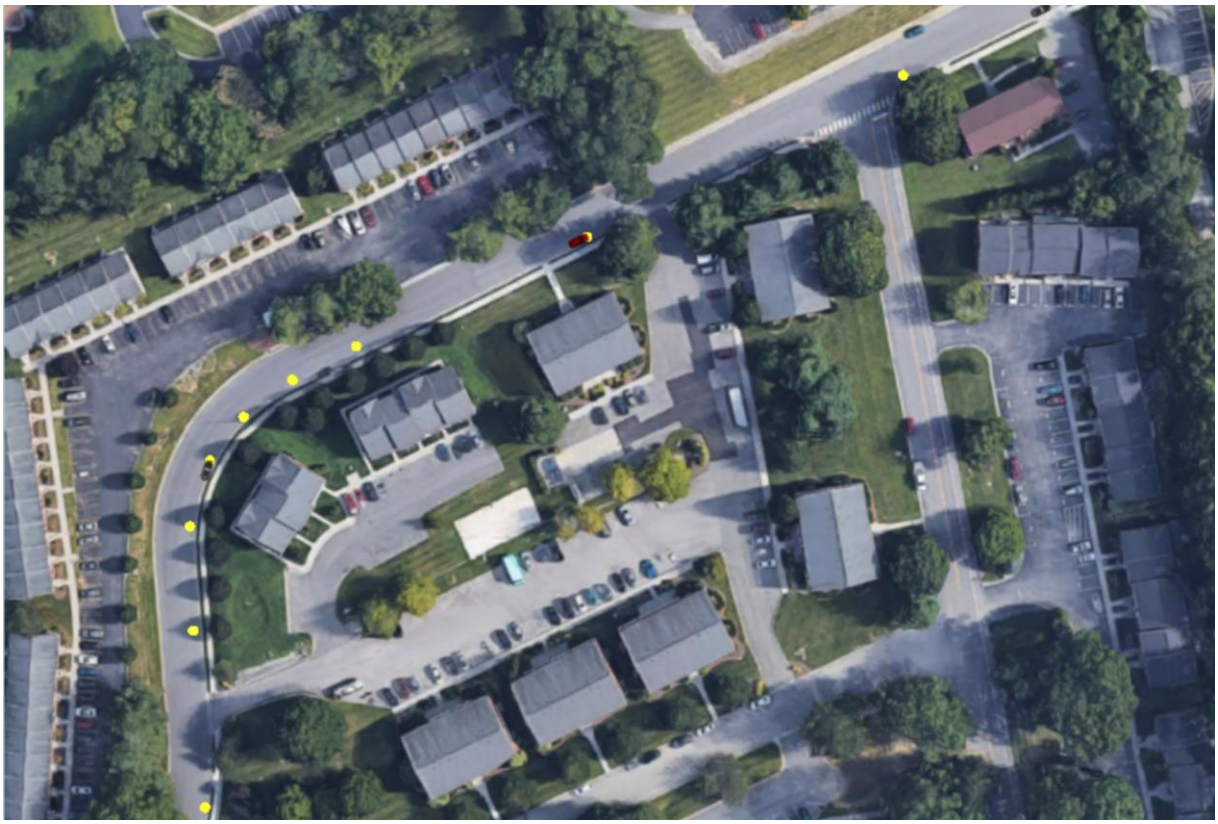

Figure 19: Move Spots on a Curvy Road

5.1.1.2 EnterPoint

Users compose their simulation models by starting from the enter points. They should be placed on the edges of the background map unless they are used to define a specific traffic component such as a parking garage or tunnel exit. Enter points are represented with the orange color in the simulation.

Enter points feed the simulation with randomly generated vehicles. In the initialization phase of the simulation, vehicle generation events are created and put into the Future Objects List (FOL). When the simulation time becomes equal to a vehicle generation event time, the event is

processed. Then, a component named as “EventFactory” schedules the enter point to generate another vehicle in the future. The vehicles start moving towards the next static object after they are generated.

Enter points generate vehicles uniformly distributed between a minimum and maximum time limit. CANVAS provides an interface for users to define the limits. In this way, users can set an enter point which generates vehicles frequently while another enter point might generate vehicles very rarely. Users can update the enter point settings by simply clicking on the points instead of recomposing the model from scratch when they need to make a change.

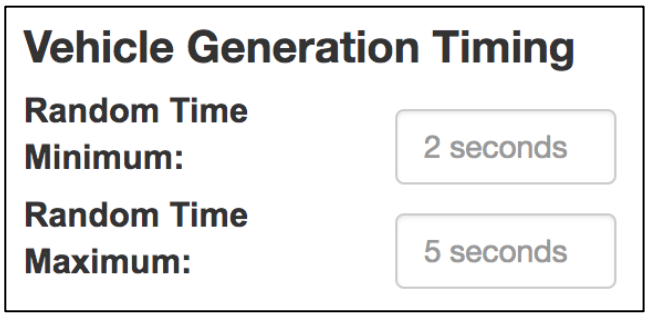

Figure 20: CANVAS Enter Point Settings

5.1.1.3 ExitPoint

Users finish composing their paths with the exit point and every vehicle leaves the simulation from an exit point. Exit times of vehicles are recorded for data collection. The black color is used to represent the exit points.

5.1.1.4 Merge

Merges have two previous static objects. This component is used when two lanes merge into one. The vehicle that becomes closest to a merge occupies the spot. Vehicles coming from the other lane checks if the merge spot is occupied by a vehicle from the other lane. If it is, they wait until the occupier leaves.

Figure 21: Basic Merge Example Screenshot

In Figure 21, two parallel lanes merge into a single one. The graph-representation of this model is presented in the following.

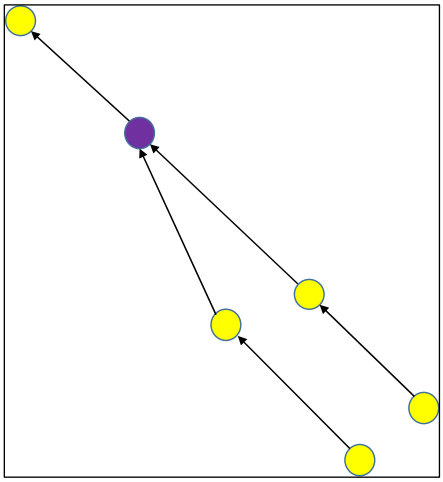

Figure 22: Basic Merge Example Graph Representation

5.1.1.5 Fork

Forks have two next static objects. When a vehicle reaches to a fork, it has to decide which way to continue. Forks are extensively used when roads intersect each other often. They also facilitate lane selection before arriving to traffic lights. A screenshot of a fork and its graph representation is provided in the following.

Figure 23: Basic Fork Example Screenshot

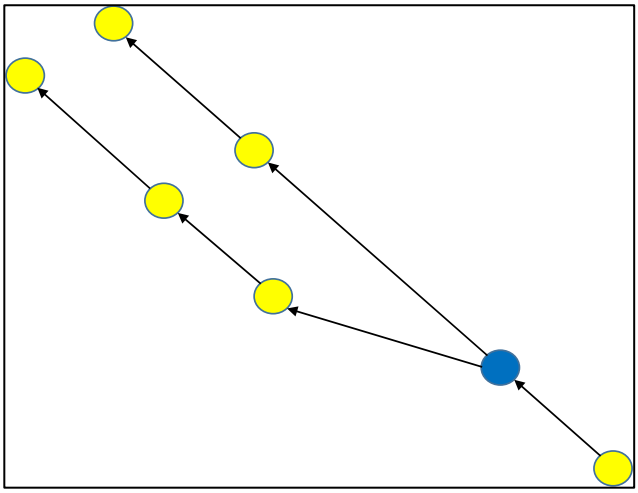

Figure 24: Basic Fork Example Graph Representation

A CANVAS fork is designed to direct vehicles to one of the next static objects with a certain probability. CANVAS provides an interface for users to set the probability when a move spot is converted into a fork. Users can update the CANVAS forks by simply clicking on the points inside the scene.

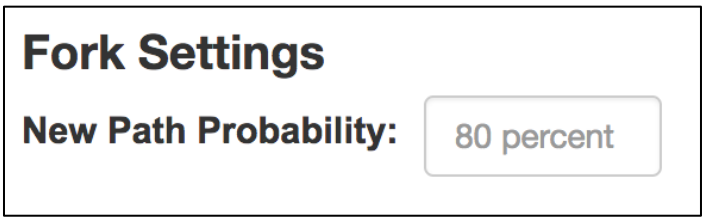

Figure 25: CANVAS Fork Settings

5.1.1.6 TrafficLight

A vehicle that arrives at a traffic light has to stop if the light is in the red state. Traffic lights function exactly same as move spots when they are in the green state. Traffic lights are created individually. Therefore, they don't know about the state of other traffic lights. For these reasons, the traffic light timing has to be set with the utmost caution. Wrong timing might cause the vehicles coming from different directions overlap with each other. The network paths intersect each other at traffic intersections without having any connection between them. Thus, the only way to prevent collisions is to utilize traffic lights with the correct time settings. In Figure 26, a traffic intersection in downtown Blacksburg, Virginia is provided. The 4-way intersection has eight different traffic lights that are completely unaware of each other.

Figure 26: A Traffic Intersection in Blacksburg Downtown

The model composer provides traffic light settings with three attributes: (1) Green Start Time, (2) Green Duration, (3) Red Duration. The user interface of the traffic light settings can be seen at Figure 9. The green start time defines the time that the traffic light will be green first time during the execution. In Figure 27, the graph representation of the intersection with numbered traffic lights is provided. An example traffic light timing of each group can be seen in Table 4. The traffic lights in the same group can be at the same state since their paths do not intersect each other. Only one group can be in the green state at a time with these settings.

Table 4. Example Traffic Light Settings in a 4-way Intersection

	Green Start Time	**Green Duration**	**Red Duration**
Group 1	0	10	30
Group 2	10	10	30
Group 3	20	10	30
Group 4	30	10	30

In the beginning of the simulation “Group 1” is in the green state since the green start time of this group is “0”. The green duration defines the time frame that this group can stay in the green state. “Group 1” switches to the red state at time “10”. This group can be green again at the time “40” since the red duration is “30”. The all groups should be designed with a similar approach so that only one group can have the green state at any time during the simulation.

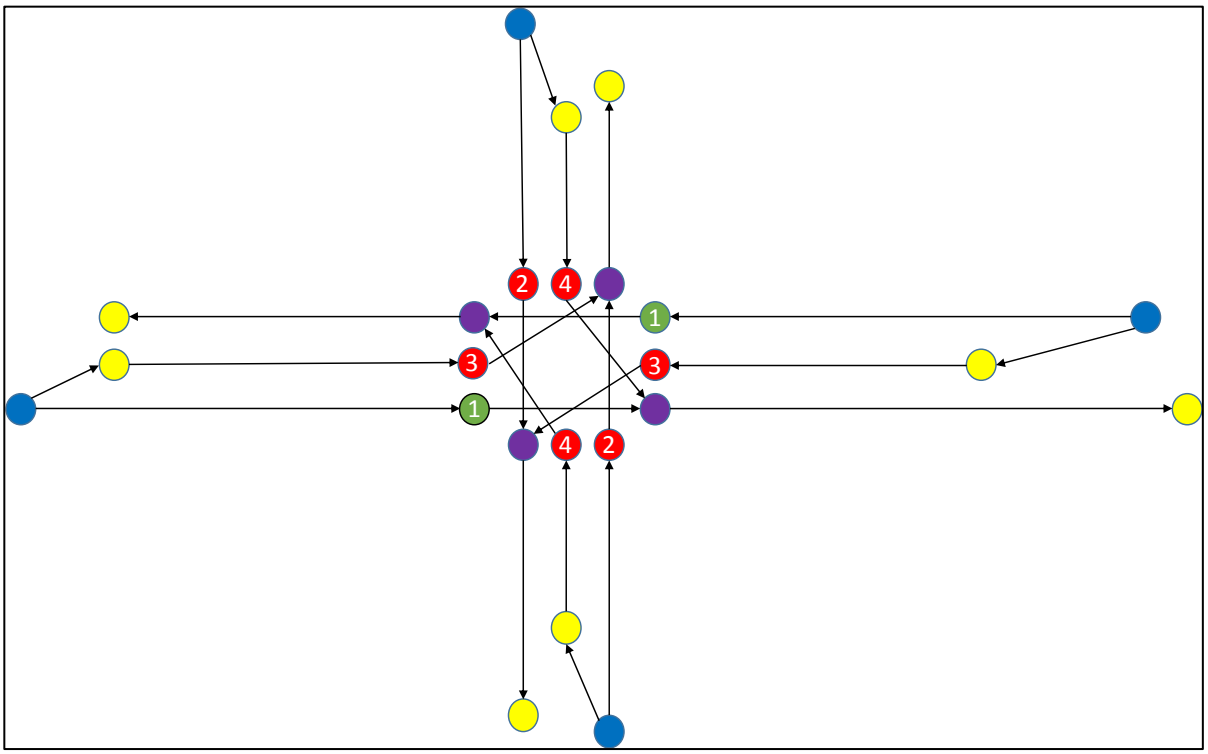

Figure 27: Blacksburg Downtown Intersection Graph Representation

5.1.2 Dynamic Objects

Dynamic objects are the moving objects in the simulation. They move on the graph generated by static objects and their connections. They change their speed and directions according to their interactions with other simulation objects. Apart from static objects, they also interact with each other. For example, if a dynamic object gets close to a slower object, it has to slow down to prevent collision. The interaction among objects is presented in Section 5.3.

CANVAS provides an extensible component architecture. Currently, only dynamic object type is "Vehicle" in CANVAS. Other object types such as pedestrians and bicycles can be created by extending the "DynamicObject" class. We only discuss the vehicle as dynamic object in this research.

5.1.2.1 Vehicle

Vehicles are the main components of the simulation visualization in which they are created at enter points and removed at exit points. There can be thousands of vehicles generated during a simulation execution. They are assigned different lengths and speeds between the fixed minimum and maximum values to provide more realistic simulation visualization. CANVAS provides automobiles as well as other vehicles such as trucks and buses. The length of vehicles plays a significant role in the conceptual framework calculations. The vehicle images used in CANVAS are provided in Figure 28.

Figure 28: CANVAS Vehicle Images

5.2 Events

Different kinds of events can be generated during a simulation execution. Some events can be generated by only one object while others can be generated as a result of object interactions. For example, a vehicle might change its speed if it arrives at a red light or if there is a slower vehicle ahead. However, a vehicle creation event can be generated only by an enter point.

All events are wrapped into JSON format and put into the message buffer before they are sent to the client to be visualized. The JSON formatted messages and their processing in the simulation visualizer is presented in Chapter 6.

5.2.1 ChangeTrafficLightState

The state of each traffic light is assigned before the simulation execution starts. If "Green Start Time" is "0", the traffic light starts with the green state. Otherwise, its state is red for the time equal to "Red Duration". A new traffic light state change event is created and put into the future event list every time a traffic light changes its state. The following method is called to generate a "Change Traffic Light State" event.

```
Event scheduleTrafficLightStateChange(trafficLight, currentTime)
{
      IF trafficLight.STATE == STATE.GREEN
            nextEventTime = currentTime + trafficLight.GREEN_DURATION
      ELSE
            nextEventTime = currentTime + trafficLight.RED_DURATION
      END IF
      return TrafficLightStateChangeEvent(trafficLight, nextEventTime)
}
```

This event is processed when the simulation execution time becomes equal to the next event time. After changing the state of the traffic light, a future state change event is generated for the same traffic light. This loop continues until the end of the simulation.

```
void processTrafficLightStateChangeEvent(trafficLight, currentTime)
{
      IF trafficLight.STATE == STATE.GREEN
            trafficLight.STATE == STATE.RED
      ELSE
            trafficLight.STATE == STATE.GREEN
      END IF
      Event = scheduleTrafficLightStateChange(trafficLight, currentTime)
      futureEventList.add(Event)
}
```

5.2.2 CreateVehicle

A vehicle creation event for each enter point is generated at the initialization phase of the conceptual framework. A vehicle is added to the active vehicles liest when the scheduled event is processed. Random generators are used to assign a speed and length to the vehicle. The scheduled event is put into the future event list to be processed in the future.

```
Event scheduleVehicleCreationEvent(enterPoint, currentTime){
      eventTime = UniformRandom(MIN_GENERATION_TIME, MAX_GENERATION_TIME)
                        + currentTime
      length = Random(MIN_LENGTH, MAX_LENGTH)
      speed = UniformRandom(MIN_SPEED, MAX_SPEED)
      vehicle = new Vehicle(length, speed)
      return VehicleCreationEvent(EnterPoint, vehicle, eventTime)
}
```

The vehicle creation event has three main components: (1) Enter Point, (2) Vehicle, (3) Event Time. The created vehicle is put into the active vehicles list from the specified enter point at the generated event time.

5.2.3 DeleteVehicle

A vehicle is removed from the simulation when it arrives at an exit point. The vehicle might have multiple connections with other static and dynamic objects. For example, there might be another vehicle just behind the vehicle. Therefore, the “next” and “prev” references must be updated when the vehicle is removed. Data collection about the vehicle also processed at this moment. For example, the deletion time is saved to calculate the average time spent in the simulation.

The “DeleteVehicle” event is processed when a vehicle arrives at an exit point. Therefore, this event does not put into the future event list.

```
void deleteVehicle(vehicle)
{
      // Remove the connections between the exit point and vehicle
      IF vehicle.prev != null
            vehicle.prev.next = NULL
            vehicle.prev = NULL
      ENDIF
      vehicleList.remove(vehicle)
      Record the deletion time // For data collection purposes
}
```

5.2.4 ChangeSpeed

Vehicles start moving with a constant speed when they leave from an enter point. However, multiple interactions might trigger vehicles to set a temporary speed. The temporary speed cannot be higher than vehicle's initial speed since there is not a mechanism to trigger the vehicle to move faster. Temporary speed can have any value that is lower than the vehicle's current speed. In many cases such as arriving to a red light, the vehicle reduces its speed to zero.

The vehicle sets its current speed back to its original speed when the bounding condition is removed. For example, the slower vehicle ahead might choose a different direction on a fork object. Since the connection between them is removed, the vehicle can move faster with its original speed.

```
void changeSpeed(vehicle, tempSpeed)
{
      IF vehicle.currentSpeed > tempSpeed
            vehicle.currentSpeed = tempSpeed
      ELSE
            vehicle.currentSpeed = vehicle.initialSpeed
      ENDIF
}
```

5.2.5 ChangeDirection

Vehicles always move on a straight path between two static objects. Therefore, they change their direction every time they arrive at a static object to move towards the new target object. The rotation is calculated according to the coordinates of the current and target static objects.

```
void changeDirection(vehicle)
{
      vehicle.rotation = calculateRotation(vehicle.currentSpot.x,
            vehicle.currentSpot.y, vehicle.targetSpot.x,
            vehicle.targetSpot.y)
}
```

5.3 Simulation Object Interactions

The events presented in Section 5.2 are generated as a result of the simulation object interactions. The state of each dynamic object is recalculated in every time frame. If a vehicle can move, its coordinates are updated with its current speed and rotation. If it arrives at a static object as a result of the coordinate update, the new state of the vehicle is calculated (e.g remove vehicle if it arrives at an exit point). Moreover, if the time stamp of the earliest event in the future event list (e.g "CreateVehicle") is equal to current simulation time the event is processed at this time frame. The simulation runtime algorithm is summarized in the following.

```
WHILE (currentTime < simulationDuration)
      WHILE the time stamp of the first event in FOL is equal to current time
            Process the Event
            Remove the event from FOL
      END-WHILE
      FOR each vehicle in active vehicles list
            IF vehicle Can Move
                  IF vehicle's current speed is zero, update it
                  Calculate vehicle's new coordinates
                  IF vehicle arrives at a Static Object
                        Calculate vehicle's new state according to the type
                        of the static object (e.g DeleteVehicle if ExitPoint)
                  ELSE IF the distance between vehicle and the vehicle ahead
                          is less than the VehicleDistanceLimit
                        Reduce vehicle speed
                  ENDIF
            ENDIF
      END-FOR
      Update currentTime
END-WHILE
```

5.3.1 Static Object to Dynamic Object Interaction

Static objects provide the structure for dynamic objects to move around. Each static object provides a different mechanism for vehicles and some of them let vehicles make decisions. For example, if the vehicle arrives at a fork, it has to choose a direction. As a result of these interactions, one of the events presented in Section 5.2 is generated.

5.3.1.1 EnterPoint – Vehicle

A vehicle is always generated at an enter point. The vehicle inherits some of the enter point's attributes when it is generated. An id, speed, and length are assigned to the vehicle at the time it is generated. The vehicle gets the rest of its attributes from the static object.

- **x:** EnterPoint.x
- **y:** EnterPoint.y
- **currentSpot:** Enter Point itself
- **targetSpot:** EnterPoint.next
- **rotation:** Rotation is calculated by coordinates of the current spot and target spot.

The vehicle's "prevDynamicObject" property is null when it is created. The algorithm that assigns "nextDynamicObject" is presented in Section 5.3.2.1. The only event that is created due to interactions between an enter point and vehicle is "CreateVehicle".

5.3.1.2 ExitPoint – Vehicle

The "ExitPoint" and "Vehicle" interaction only generates the "DeleteVehicle" event. When a vehicle arrives at an exit point the vehicle is deleted with the algorithm presented in Section 5.2.3.

5.3.1.3 MoveSpot – Vehicle

When the vehicle arrives at a move spot, its target static object is updated with the "next" value of the move spot. Rotation of the vehicle must be changed since it starts moving towards to a different object. This interaction generates a "ChangeDirection" event.

5.3.1.4 Merge – Vehicle

Vehicles can arrive at merges from two different directions since merges have two previous static objects. Therefore, a vehicle must check if a vehicle from the other direction is already occupying the merge. This control is made when the algorithm checks if a vehicle can move. If the following algorithm returns false, the vehicle's current speed is set to zero temporarily. If the algorithm returns true and its speed already has reduced to zero, the speed is updated back to its initial value.

```
boolean canMoveOnMerge(merge, vehicle)
{
      IF merge has an occupier
            IF merge.occupierId != vehicle.id //If occoupier is
                                              //another vehicle
                  return false
            ENDIF
      ENDIF
      return true
}
```

As in the MoveSpot – Vehicle interaction, the vehicle has to change the rotation when it arrives at a merge. Therefore, this interaction also generates a "ChangeDirection" event.

5.3.1.5 Fork - Vehicle

The vehicle that arrives at a fork must choose a direction to move. The direction is chosen with the fork's direction probability. The direction selection probability of a fork is set by users in terms of percentage when the fork object is created in the model composer.

```
setTargetStaticObject(vehicle, fork)
{
      Choose a randomValue between 0 and 100
      IF randomValue <= fork.mainPathProbability
            vehicle.targetSpot = fork.next
      ELSE
            vehicle.targetSpot = fork.nextAlternative
      ENDIF

}
```

After the target spot is selected, a "ChangeDirection" event is generated to let the vehicle update its rotation.

5.3.1.6 TrafficLight – Vehicle

Traffic lights act exactly same as move spots when they are in the green state, but they block vehicles when their state is red. When a vehicle arrives at a traffic light, the state of the traffic light is checked. If it is red, the vehicle's speed is set to 0 temporarily. The vehicle's speed is set to its initial speed when the traffic light switches back to the green state. As a result of the TrafficLight – Vehicle interactions, the "ChangeSpeed" and "ChangeDirection" events are generated.

5.3.2 Dynamic Object to Dynamic Object Interaction

Dynamic Objects also interact with each other. Currently, the only subclass of "DynamicObject" in CANVAS is "Vehicle". The interaction among vehicles is discussed in the following subsection.

5.3.2.1 Vehicle to Vehicle Interaction

There might be several vehicles between two static objects. Therefore, there should be a mechanism to provide the interaction among vehicles moving between the two static objects. The vehicles between the two static objects form a doubly linked list. Thus, each vehicle has a reference to the vehicle ahead and behind. In this way, we can calculate results of their interactions in O(1) time.

The doubly linked list is generated by assigning references to the next and previous objects. Therefore, the linked list structure does not use an extra memory in terms of Big O notation. When the vehicle arrives at its target static object, its connections are removed and new connections are established via its new target spot. All static objects except "ExitPoint" provides the same functionality. In the following, the algorithm that updates the vehicle's connections is presented.

```
void vehicleDidArriveAtStaticObject(vehicle)
{
      // Update the references between the vehicle and the static objects
      // Set the occupier id of the static object
      vehicle.currentSpot.occupierId = vehicle.id

      // Previous vehicle is the new incoming object
      vehicle.currentSpot.incomingDynamicObj = vehicle.prev

      // Set the static object as the current spot of the vehicle
      vehicle.currentSpot = vehicle.targetSpot

      // Set the target object of vehicle via the new current spot
      vehicle.targetSpot = vehicle.currentSpot.next

      // Remove the connections with the vehicle behind
      IF vehicle.prev != NULL
            vehicle.prev.next = NULL
            vehicle.prev = NULL
      ENDIF

      // Find the vehicle that is closest to the target spot
      vehicleAhead = vehicle.targetSpot.incomingDynamicObj

      // Find the vehicle at the end of the linked list
      WHILE(vehicleAhead.prev != NULL)
            vehicleAhead = vehicleAhead.prev
      END-WHILE

      // Connect vehicles
      vehicle.next = vehicleAhead
      vehicleAhead.prev = vehicle
}
```

The Vehicle-to-Vehicle interaction is specifically required since the vehicle needs to change its speed temporarily in case it becomes close to the vehicle ahead. If the vehicle ahead is moving with a slower speed, its temporary speed is set to current speed of the vehicle ahead. Therefore, they continue moving with same speed. The connection between vehicles are removed when the vehicle ahead reaches to the next static object. If they continue moving towards the same direction, a new connection is established which might let the vehicle reduce its speed again.

Chapter 6: CANVAS Implementation

The component-based structure of CANVAS can be categorized under two main groups: the Client-Side and Server-Side. Users interact with the client-side to compose their simulation models and visualize the simulation results. What happens under the hood is completely hidden from the client-side since all simulation execution runs on the server. The execution on the server and the visualization on the client significantly increase the complexity of the environment. Moreover, maintainability and extensibility of the software require higher efforts since an update on the server-side must be matched on the client-side. On the bright side, CANVAS is enabled to serve concurrent users who use different kind of devices thanks to this development strategy.

The CANVAS components are designed to perform a single task. The connection among components on the server side is provided by Java function calls while the communication between the client and server-side components are facilitated by WebSockets and JSON messaging. The server is designed to serve many users at the same time, and usage of computer resources are limited for each user. Therefore, the server cannot be overwhelmed by a user who executes an intensive simulation.

6.1 Model Composition

Simulation developers must have programming knowledge to compose models in many existing simulation environments. Creating new model components or updating the existing components must be conducted by expert simulation developers in many tools. CANVAS is designed to serve non-expert simulation developers by providing well-tested and readily available model tools. A CANVAS simulation model can be updated by changing the parameters of the existing components or simply adding new components through the model composer instead of making changes to the code. Therefore, users do not need to have programming knowledge or contact to an expert to make use of CANVAS.

CANVAS is made available within standard web browsers for simulation developers to use from anywhere. Existing web standards are used to develop CANVAS providing a plugin-free and platform independent simulation IDE. Technologies such as XHTML, CSS, and JavaScript are used to create the web pages of the IDE. The Three.js library has played a significant role to manipulate the HTML canvas element, which provides the display of the simulation model. JavaScript enables the connection between the model composer and HTML canvas element during the model composition.

Users create a project with an aerial view of a traffic network. The view might be a photograph or more likely a satellite image. There are many free tools to get a view of the desired area. CANVAS users can get an aerial view of almost any location on the earth by using Google Maps [2018]. Google Maps has the option of turning off any labels on the satellite view. Therefore, a plain image of areas can be obtained to be used as a background map in CANVAS. In Figure 29, Simulator.xhtml page is presented with an aerial view of an intersection in Blacksburg, Virginia.

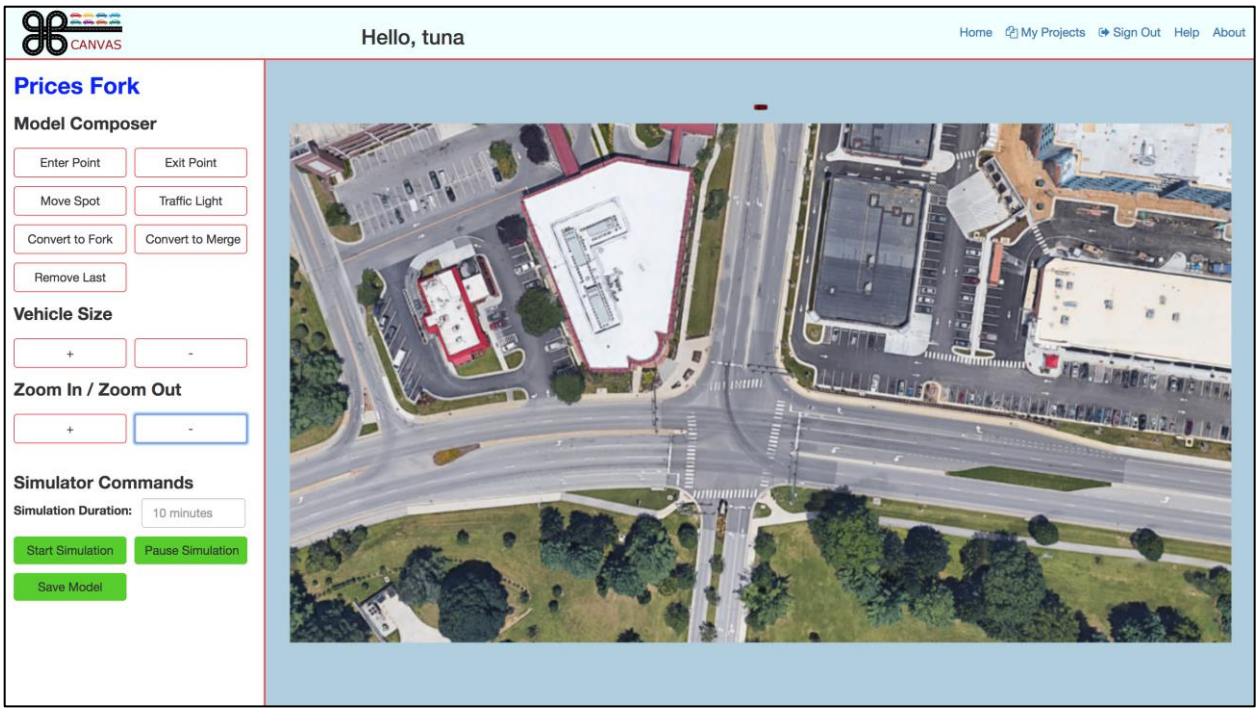

Figure 29: A CANVAS Project with Aerial View of an Intersection in Blacksburg, VA

The control panel that includes the model composer and simulation settings facilitates the user interactions with the IDE. The panel is dynamically updated to ask for parameters in the model composition process. For example, an inner form is displayed when a user clicks to the "Traffic Light" button as it can be seen in Figure 9.

CANVAS tracks user clicks within the browser. The selected simulation object is added to the simulation model when the user clicks on the map. Enter points, exit points, move spots and traffic lights can be directly added to the simulation model while forks and merges have to be converted from a move spot. If the user does not enter the required parameters, the selected object cannot be added to the model. Users can remove the objects starting from the last added object. A random object cannot be removed since the graph structure cannot be preserved when a node is removed.

CANVAS supports traffic network simulations on any scale. Size of vehicles must be adjustable to provide accurate visualization on any aerial image. CANVAS displays a vehicle image on the top of the map for users to preview a standard automobile. Users can change the size of the vehicle on the control panel to define the appropriate vehicle size for the provided background image. The size of the vehicle is sent to the simulation manager as a parameter within the simulation model. The constants of the simulation algorithm are generated according to the vehicle size. For example, speed of vehicles is proportional with their size.

The model composition is fully processed on the client-side through JavaScript. The simulation model must be transferred to the server for the simulation execution. JavaScript Object Notation is used to wrap the data in CANVAS. This data is unwrapped in simulation builder to prepare a

traffic network graph on the server. Figure 30 presents a basic simulation model that has only three static objects.

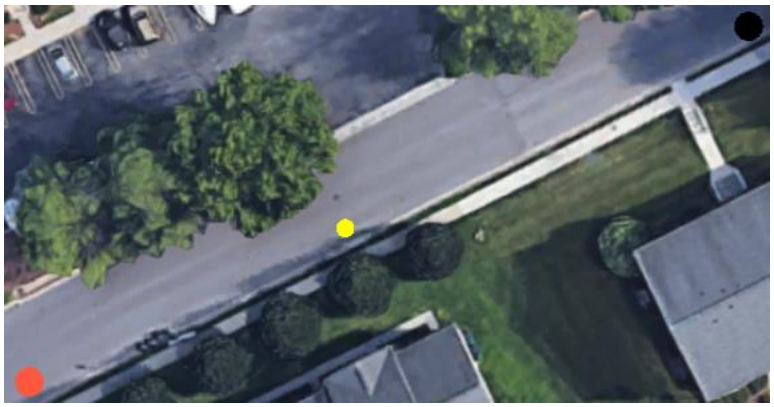

Figure 30: A CANVAS Model with Three Static Objects

The JSON representation of a model is created when a user clicks to the "Start Simulation" button. The model is wrapped into a JSON array before sending it to the server for execution. The server-side execution and event visualizations are discussed in the following sections.


```
[{"id":"s1","type":"EnterPoint","x":"-388.8564","y":"13.6854","nextId":"s2"},
{"id":"s2","type":"MoveSpot","x":"-214.248","y":"70.31","nextId":"s3","prevId":"s3"},
{"id":"s3","type":"ExitPoint","x":"8.4944","y":"180.7427","prevId":"s2"}]
```


Figure 31: JSON Representation of a Basic Model

6.2 Model Storage and Collaboration in CANVAS

Many existing simulation environments are developed under the SaaP paradigm which makes collaboration with other developers difficult. Simulation developers can use many of the existing tools only from a single device. CANVAS provides tools to store and share simulation models. These tools increase the productivity and reduce the overall development efforts for simulation models. Users can copy other users' models and make changes according to their needs.

Simulation models can have hundreds of components that are connected to each other. Thus, composing a simulation model can be an exhausting and time-consuming process. A simulation developer might need a break and want to continue model development later. Moreover, a simulation developer might develop the model on different devices. For these reasons, the storage and retrieval of the simulation models are extremely important. Creating a new simulation model which is similar to an existing model is easier thanks to model storage. Users can copy and modify an existing model instead of composing it from scratch.

The model storage and retrieval are not easy tasks since CANVAS is a traffic network simulation IDE which visualizes the objects according to their coordinates. Computer monitors have different resolutions and width/height ratios. Therefore, an object coordinate can point to a completely different location in another monitor. CANVAS tackles these challenges gracefully. First, a model might not fit into another device's screen automatically since monitors have different ratios. CANVAS provides a tool for users to zoom in and zoom out to the simulation model. Users can zoom out the simulation model until whole background map is displayed when it does not fit into a screen. Second, coordinates of the objects cannot be directly saved into the database since the coordinates are determined according to resolution of the screen. The model storage tool sends object coordinates and the background map size into a function to calculate the object locations relatively. When a simulation developer uses the same model on another device, the values in database are sent into another function to calculate the new coordinates.

The model storage also facilitates collaboration with other developers. Users can make a project public which makes the project be seen by all CANVAS users. After a project is made public, another developer can copy it into his own repository and make changes to the model. The updated model can be saved under a different project name. Users also can share the models after their modifications. In this way, CANVAS encourages users for higher collaboration. Moreover, first time users can make use of existing models to learn how to develop complicated models.

6.3 CANVAS Server

The client interacts with two main server components: "Simulation Builder" and "Simulation Manager". The simulation builder provides model validation and generates a server-side graph structure as an input for the simulation manager. The simulation execution is processed by the simulation manager. The server responsibilities are discussed in the following sections.

6.3.1 Model Validation

The CANVAS model validation makes sure that a valid weighted directed graph of static objects is created. If an object is composed incorrectly, the simulation builder informs the client with an error message including the object number. The client shares the responsibility of model validation for simple cases such as missing parameters. The detailed controls are made in the simulation builder.

The model validation makes multiple controls on the graph structure and its components. First, a CANVAS simulation model must have at least one enter point and exit point. Second, references of the neighbor objects cannot be null for any static object. For example, if a fork is created, the simulation builder has to be sure that it has two valid next objects. However, model validation does not guarantee that an accepted model always generates correct results. For example, if the traffic light timing in an intersection is set incorrectly, dynamic objects might overlap within the scene. Model validation cannot make controls between the objects placed in different paths of the graph.

6.3.2 Simulation Execution

Performance and scalability are the two main concerns of a cloud-based simulation software. A special effort has been spent on the threading model of CANVAS. The client-side is in charge of the execution request since the server does not know about the state of the client. Therefore, the simulation execution is designed in a way that it only runs as long as the client requests new events. The server resources are not used when an execution thread is paused enabling other clients to utilize the resources. The visualization occurs in real time which is much slower than the execution on the server. The CANVAS threading model makes sure that a client cannot stress the server since it has to consume the generated events in real time before requesting new events.

Users are connected to the CANVAS server through WebSockets. A simulation manager instance is assigned to each client after a WS session is started. The server has a simulation session handler class that manages the connection of each client who has a unique session id. These instances are kept in a hash map in which the key is the unique session id. Therefore, when a client makes a new request to the server, the CANVAS server can identify the correct simulation instance in O(1) time.

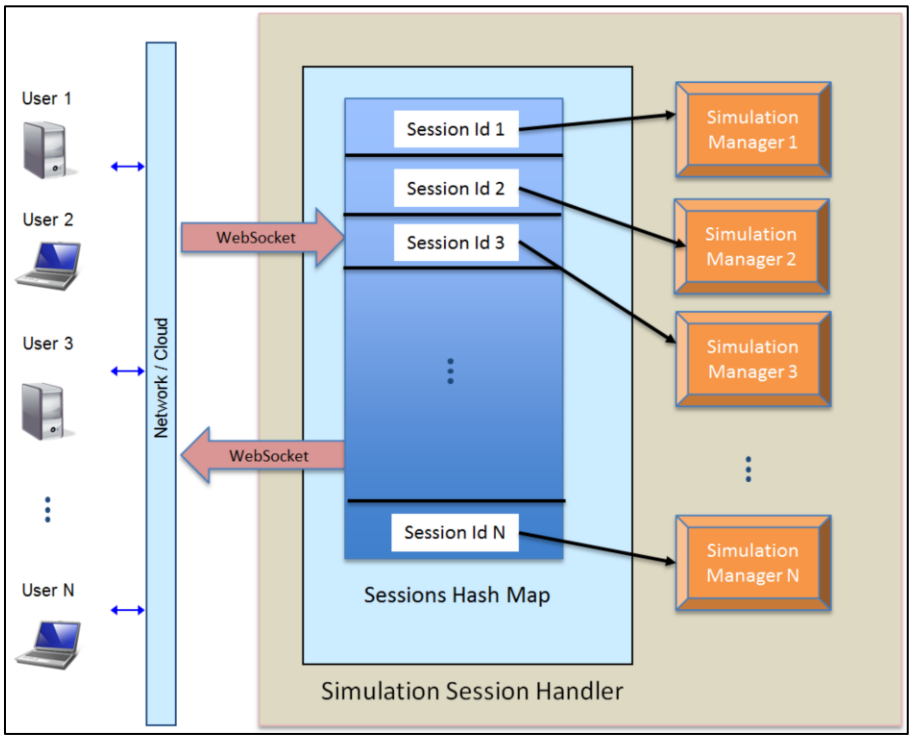

Figure 32: CANVAS WebSocket Sessions

The simulation execution is conducted by a simulation manager instance. Validated simulation models are sent to the simulation manager to prepare the traffic network graph that is the core of CANVAS conceptual framework. The events produced by the simulation manager is sent to the message manager to store them in a time-orderly buffer.

Thread safety is one of the utmost priorities among simulation executions. Static objects of Java language cannot be utilized for the conceptual framework parameters since a change on a parameter would affect the execution in another thread. To illustrate for CANVAS, a client might run a simulation instance in a very small area while another client runs a big scale simulation at the same time. Vehicles appear bigger in the small area since the size of the visualization scene is fixed in web browsers. The simulation constants (e.g speed of vehicles) have to be different in these two instances since the coordinate change of a big vehicle has to be higher than a small vehicle in another simulation execution. Therefore, local constants are generated for each simulation instance even though this approach increases the complexity.

A CANVAS client requests data when the client-buffer has low number of events to process. The request is processed by a separate thread every time the client makes a new request. The simulation execution instance of the client is found through sessions hash map for each request. The message request thread triggers the corresponding message manager to send number of events to the client. If the buffer in the message manager is empty, the thread waits until the simulation manager generates more events. After events are sent to the client, the message manager thread is terminated.

6.4 Simulation Visualization

The simulation visualization starts when events are received from the server. The visualization manager updates the display in every time frame by moving the dynamic objects according to their speed and rotation. The traffic light is the only updated static object type during the visualization. They switch between green and red colors during the visualization, giving the clear representation of their current state. The earliest event in the buffer is processed before updating the scene when the visualization time becomes equal to its time stamp.

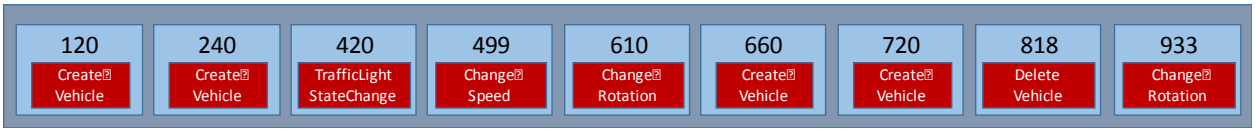

Figure 33: An Example Set of Events in Visualization Manager Event Buffer

CANVAS utilizes the Three.js library as the main visualization technology. Three.js provides simple and efficient object management within the HTML canvas scene. Three.js renderer renders the page 60 times in a second which enables updating object coordinates smoothly. The dynamic object coordinates are updated according to their speed and directions in every 1/60 seconds. They are removed from the scene when “DeleteVehicle” message is processed on the visualization manager.

The visualization manager is a lightweight component of CANVAS architecture. While the CANVAS algorithms run on the server-side, the visualization manager only creates, removes objects and update their coordinates. In this way, CANVAS is able to provide a powerful simulation visualization to thin clients. The CANVAS events include object id that is used to find the referred object. The object is simply updated with the event data sent by the server. Simulation visualization continues until receiving the “End of Simulation” message. If event buffer gets empty before the end of the simulation, visualization pauses until getting the batch of events from the server.

Chapter 7: Case Studies

This chapter presents case studies describing three different simulation models with their components. Each study explains how models are created step by step. The case study models can be created at http://orca.cs.vt.edu/canvas/ by following the instructions provided below.

7.1 Case Study 1

In this study, we created a project named as "Case Study 1" displaying a relatively small area as the background map. The background map is placed at the center of the visualization area within the web browser. The size of the map is determined according to the screen resolution automatically.

Figure 34: Case Study 1 Initial Project Screen

The HTML canvas element has extra space that can be utilized by the background map. "Zoom In" tool is used to make the background map bigger.

An automobile image is displayed on the top of the map to inform users about the standard vehicle size in the simulation. Since the displayed area is relatively small, vehicles have to appear bigger. We increase the "Vehicle Size" in the control panel. The updated background map and vehicle size are displayed as in Figure 35.

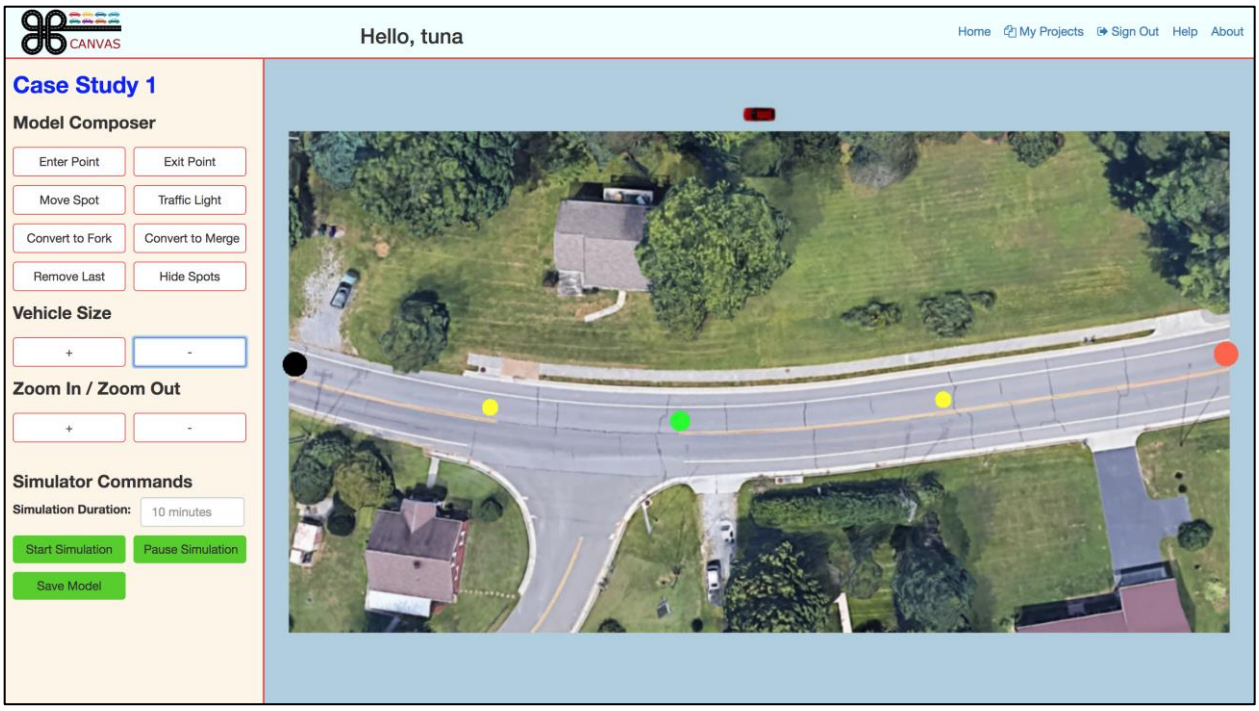

Figure 35: Case Study 1 Simulation Model

We consider a simple simulation model that includes a traffic light, enter point, exit point, and two move spots in this case study. The model composition always starts with adding an enter point in CANVAS. The composition steps are presented as follows.

1. The first component is added by clicking to the right edge of the map after the "Enter Point" is selected in the model composer. The model composer is dynamically updated requiring an input for the enter point settings as in Figure 20. The following parameters are added to the enter point.
    - Random Time Minimum: "10 seconds"
    - Random Time Maximum: "30 seconds"
2. The "Move Spot" component is selected in the model composer and added to the left-side of the enter point.
3. The "Traffic Light" component is selected. The required parameters are provided as in the following.
    - Green Start Time: "0"
    - Green Duration: "50 seconds"
    - Red Duration: "10 seconds"
4. The second move spot is added.
5. The exit point is added to the left edge of the map.

The composed simulation model is saved by clicking the "Save Model" button. The visualization is started after the simulation duration is set as "10 minutes". Vehicles are generated and moved

within the model for ten minutes. The simulation results are displayed when the last vehicles left the model from the exit point.

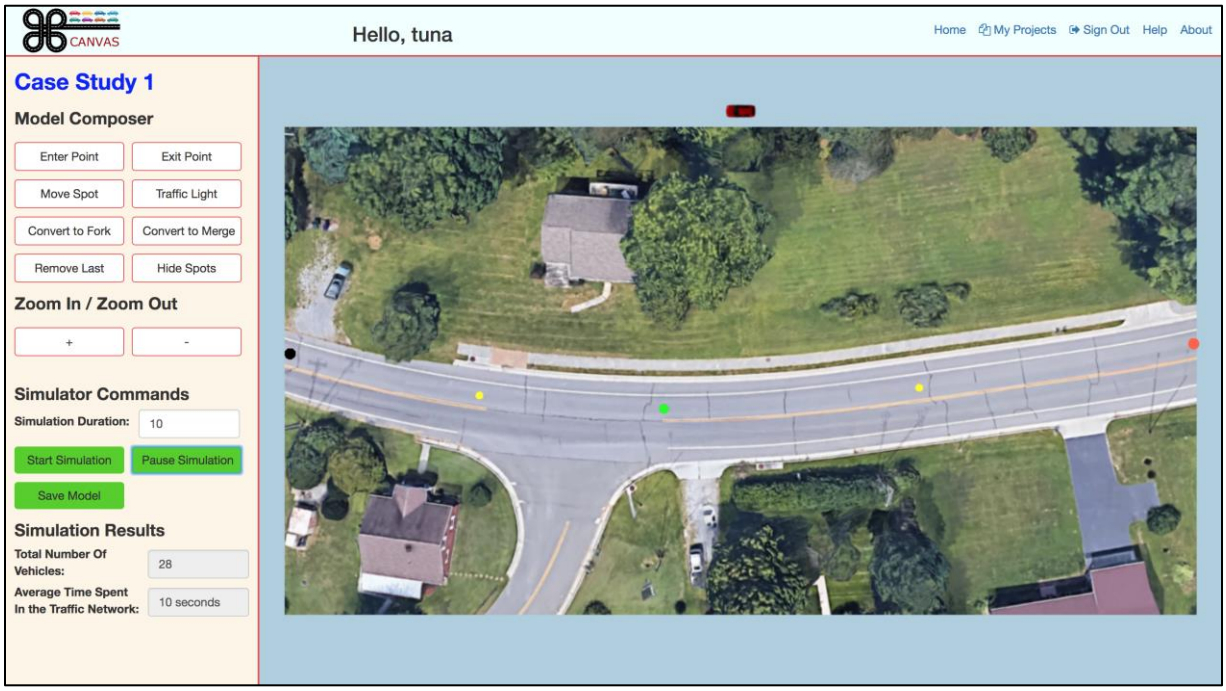

Figure 36: Case Study 1 Simulation Results

7.2 Case Study 2

This case study focuses on converting the move spots into forks(blue spot) and merges(purple spot). The composed “Case Study 2” simulation model is provided in Figure 37.

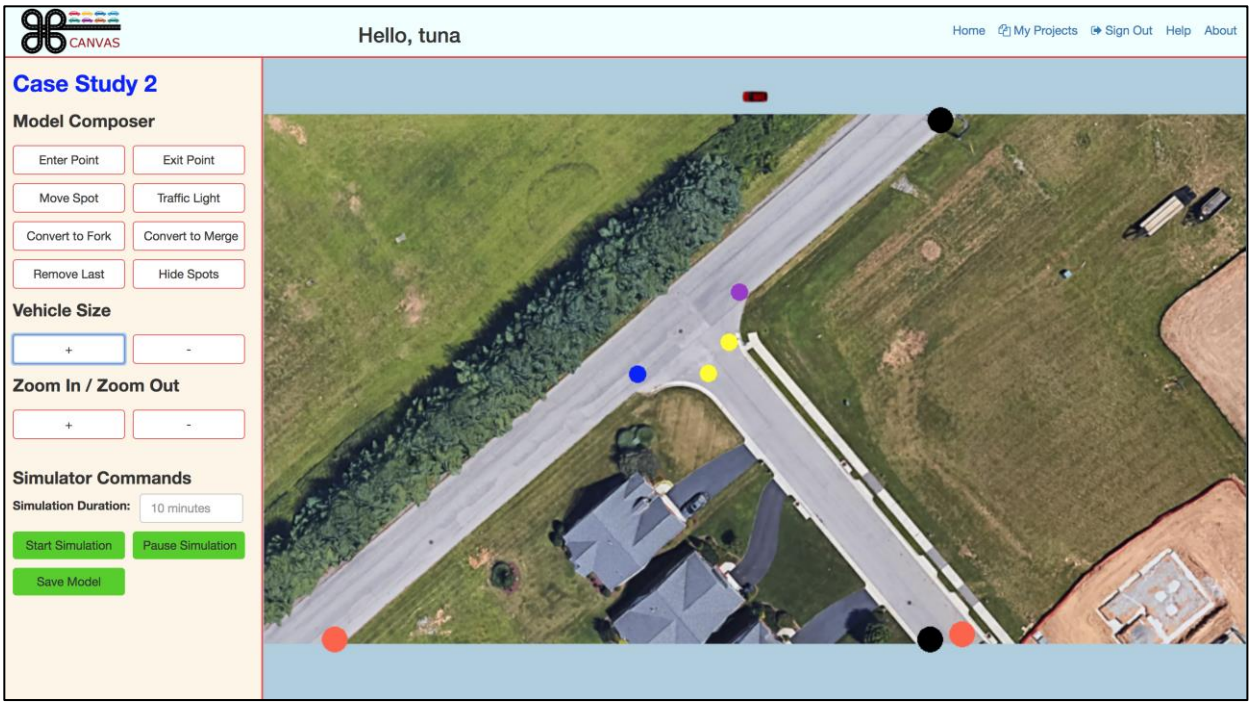

Figure 37: Case Study 2 Simulation Model

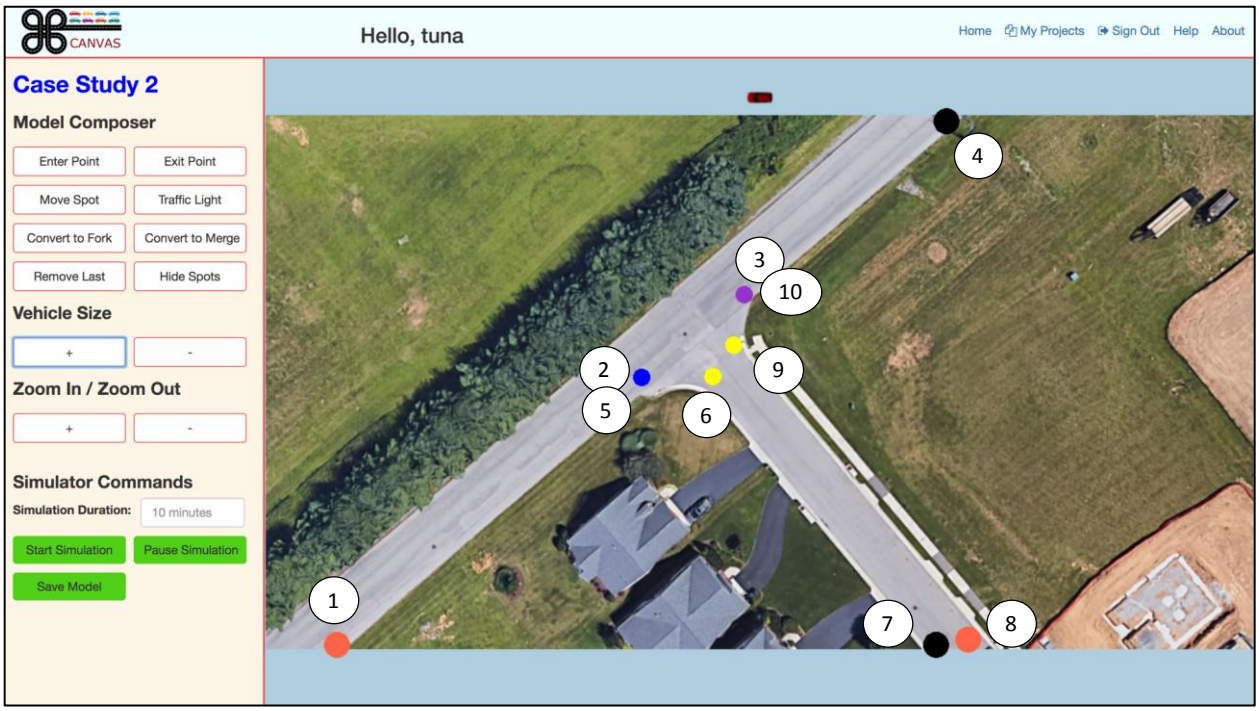

Figure 38: Case Study 2 User Interactions

Figure 38 presents user interactions with the simulation model. The composed model is created in ten steps which are described as follows:

1. Add an enter point by setting the “Random Time Minimum” as “3 seconds” and “Random Time Maximum” as “10 seconds”.
2. Add a move spot.
3. Add a move spot.
4. Add an exit point.
5. Select the “Convert to Fork” option in the model composer. Click on the move spot that is added with action 2. Enter the “New Path Probability” as “30 percent”.
6. Add a move spot.
7. Add an exit point
8. Add an enter point by setting the “Random Time Minimum” as “5 seconds” and “Random Time Maximum” as “20 seconds”.
9. Add a move spot.
10. Select the “Convert to Merge” option in the model composer. Click on the move spot that is added with action 3.

The simulation is executed for ten minutes producing 138 vehicles that spent 11 seconds on average in the traffic according to the simulation results. Figure 39 displays a moment from the visualization.

Figure 39: A Screenshot during the Case Study 2 Visualization

7.3 Case Study 3

This case study presents a complex traffic network with several components as provided in Figure 40.

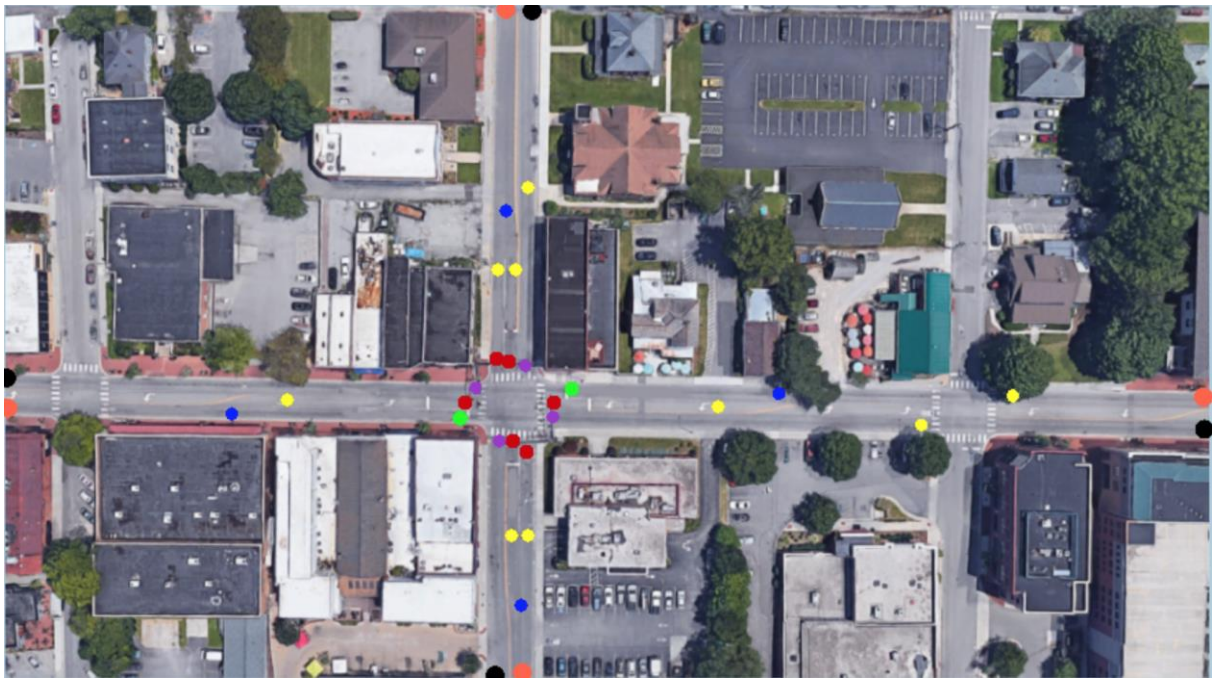

Figure 40: Case Study 3 Simulation Model

The initial version of the model is composed without forks and merges defining the main paths as in Figure 41. Each number in the figure defines a separate lane. The paths are created by starting from the enter points and ending at the exit points. Table 5 presents the random time minimum and maximum values for each enter point.

Table 5. Case Study 3 Enter Point Objects Random Time Limits

	Random Time Minimum	Random Time Maximum
Path 1 Enter Point	1 second	5 seconds
Path 2 Enter Point	2 seconds	6 seconds
Path 3 Enter Point	3 seconds	8 seconds
Path 4 Enter Point	4 seconds	10 seconds

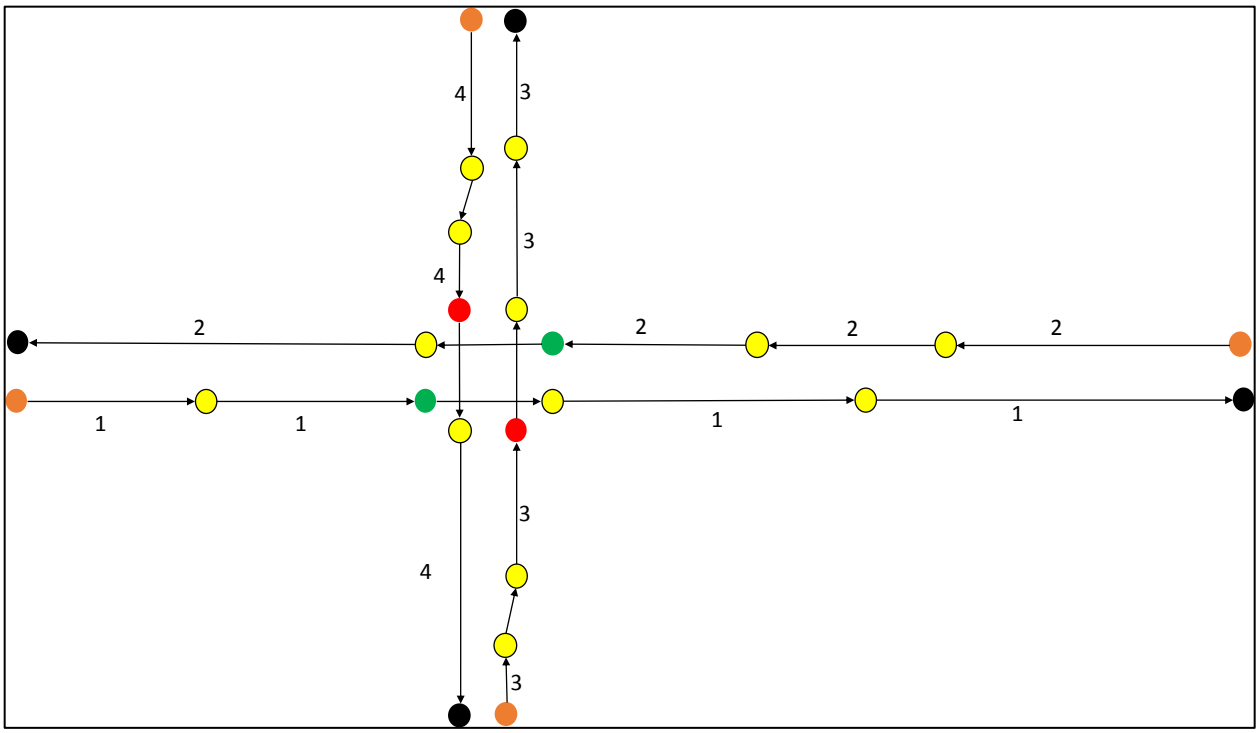

Figure 41: Case Study 3 Simulation Model Initial Graph Representation

The model is improved by converting some of the move spots to merges and forks. The converted forks start new lanes that are presented in Figure 42. The main paths are defined with the letter “A” and the new paths are defined with the letter “B” starting from the fork of each lane. The new path probability of each fork is set as in Table 6.

Table 6. Case Study 3 Fork Objects New Path Probability

	The New Path Probability
Path 1 Fork	20 percent
Path 2 Fork	30 percent
Path 3 Fork	35 percent
Path 4 Fork	45 percent

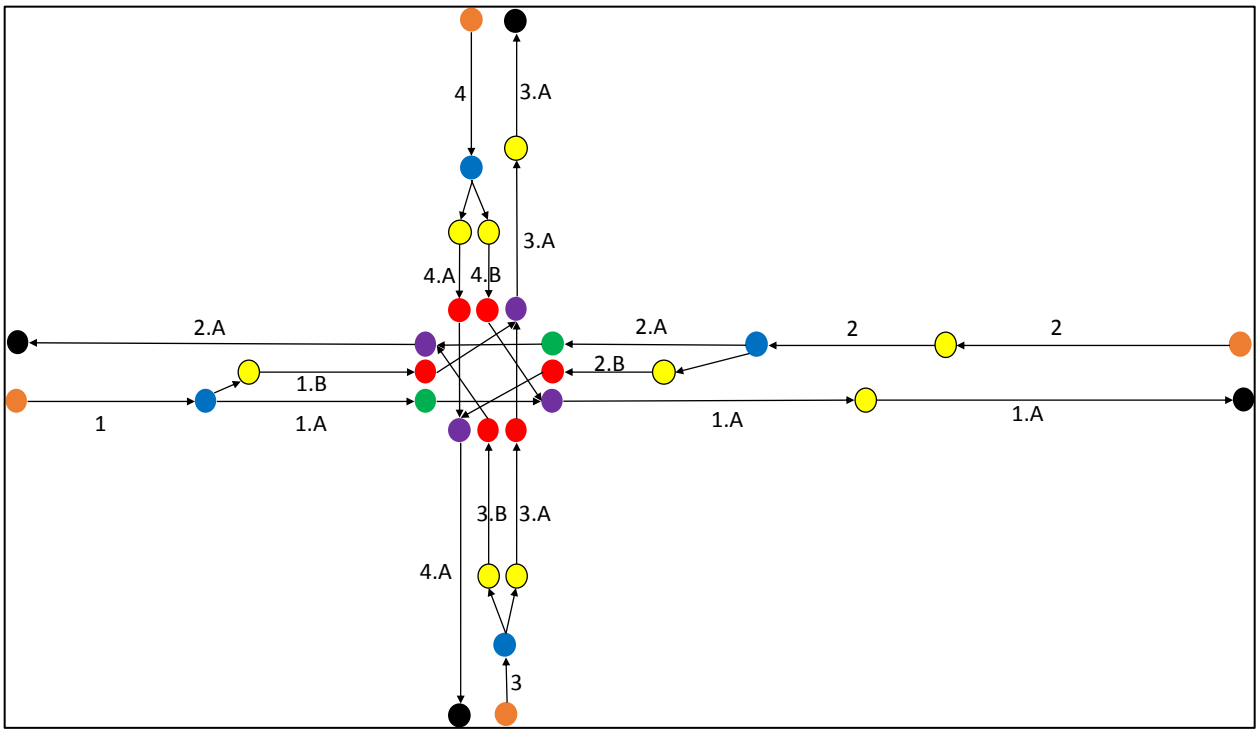

Figure 42: Case Study 3 Simulation Model Graph Representation

In a CANVAS simulation model, traffic lights on the non-intersecting paths can be in the green state at the same time. Only two traffic lights can be in the green state at a time in this model. Table 7 displays the traffic light time settings. The traffic lights in the group 1 and group 2 stays in the green state for twenty seconds while group 3 and group 4 stays in the green state for ten seconds in a minute. In this way, the timing circle is completed at the end of every minute.

Table 7. Case Study 3 Traffic Light Timing

	Green Start Time	Green Duration	Red Duration
Traffic Light 1.A (Group 1)	0	20	40
Traffic Light 2.A (Group 1)	0	20	40
Traffic Light 3.A (Group 2)	20	20	40
Traffic Light 4.A (Group 2)	20	20	40
Traffic Light 1.B (Group 3)	40	10	50
Traffic Light 2.B (Group 3)	40	10	50
Traffic Light 3.B (Group 4)	50	10	50
Traffic Light 4.B (Group 4)	50	10	50

We run the simulation model composed for Case Study 3 for ten minutes. 504 vehicles are created during the simulation duration. The average time spent in the traffic was 81 seconds.

Figure 43: A Screenshot from Case Study 3 Simulation Visualization

7.4 Case Study 4

CANVAS can simulate complex traffic networks that consist of multiple traffic intersections. In this case study, we provide a simulation model for a traffic network in Fairfax, Virginia. Figure 44 displays the composed model that has 19 enter points and 267 static objects in total.

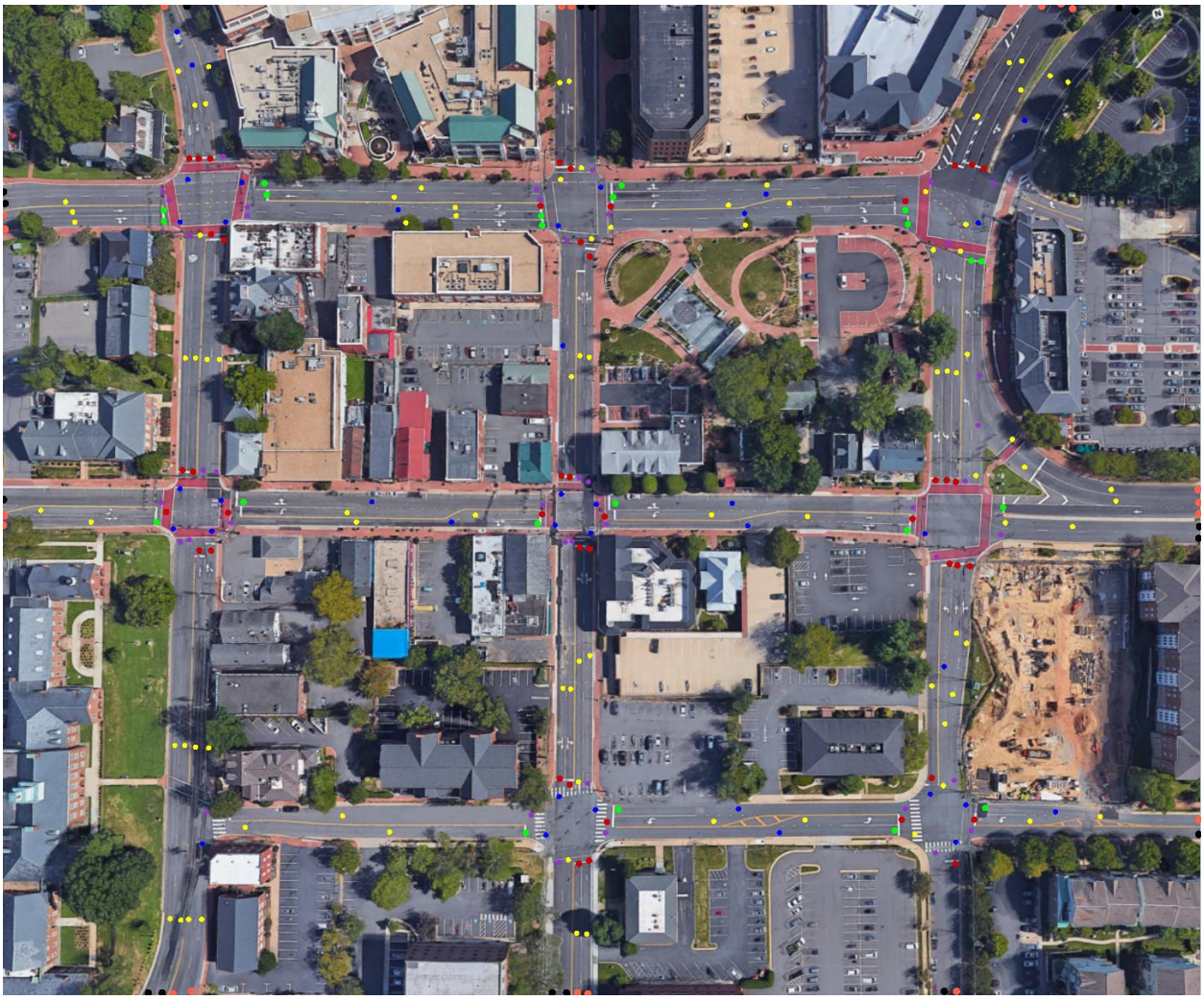

Figure 44: Case Study 4 Simulation Model

Each intersection is carefully composed as explained in Section 7.3. We set the simulation duration as 30 minutes. Vehicle generation is stopped after 30 minutes and the execution continued until the last vehicle leaves the model. Enter points, forks, and traffic lights are set with different parameters that represent a realistic behavior. Figure 45 and Figure 46 display the state of simulation at different times during the simulation visualization.

Figure 45: A Screenshot from Case Study 4 Simulation Visualization at 10th Minute

Figure 46: A Screenshot from Case Study 4 Simulation Visualization at 20th Minute

Case Study 4 simulation execution generated 2938 vehicles in total. Vehicles spent 470 seconds on average in the traffic network. Visualizing a complex model for a long time requires high number of events. According to server logs, 258968 events are generated to provide the visualization in this case study. Since events are consumed very quickly, the client made event requests much more often that the previous case studies.

Chapter 8: Conclusions and Future Research

8.1 Conclusions

CANVAS eases model development and simulation visualization for non-expert users. The readily available components can be used to define a simulation model without requiring any programming knowledge. The research described herein shows that a cloud-based visual simulation IDE can provide advanced visualization within web browsers.

The collaboration system of CANVAS reduces the overall development effort by enabling users to share their simulation models. Most importantly, a simulation model can be visualized on different computers. CANVAS stores and retrieves simulation models by considering the screen resolution of the computer on which visualization is done.

The component-based software architecture enables CANVAS to be used by multiple users concurrently. Server resources are used only as long as clients consume the data. This approach prevents any extensive usage of the resources enabling new clients to use the resources immediately.

We use traffic networks as the problem domain in which to develop a cloud-based visual simulation environment. The CANVAS design, asynchronous visualization protocol, and the IDE features can be applied to any other problem domain. Even though it is not the main goal of the research described here, the CANVAS conceptual framework provides a realistic representation of traffic networks. The algorithms and graph-based model representations can be further improved for a better traffic network conceptual framework.

8.2 Future Research

CANVAS provides a set of components to compose simulation models. The model components can be improved as follows:

1. Make minimum and maximum speed parameters modifiable by users.
2. A vehicle can choose one direction among three options in a real traffic network. Currently, CANVAS only provides two options for forks. “Fork” static objects can be extended to have a third option.
3. Vehicles on main roads have priority to move before vehicles coming from side roads at traffic merges. Currently, CANVAS merges work first-come-first-move basis. “Merge” static object can have an optional parameter to provide priority to one of the lanes.
4. In CANVAS, vehicle speed is updated in a single time frame which is unrealistic. CANVAS “Vehicle Speed Change” can be improved to change the speed gradually. Therefore, users can feel the vehicle acceleration.

The research described herein focuses on the visualization capabilities of the IDE. The data collection modules are limited to providing only two results to the users: “Number of Vehicles Created” and “Average Time Spent in the Traffic Network”. A more detailed data collection module can be developed.

The asynchronous visual protocol is presented as our methodology to serve multiple users concurrently. The client data consumption is limited to prevent extensive usage of the server resources. However, this research does not provide any performance metrics. The protocol variables such as number of transferred messages per request can be optimized after running multiple performance tests. Improvements on the asynchronous visualization protocol can provide better scalability, allowing a higher number of clients to run simulations on CANVAS concurrently.

REFERENCES

Adobe (2018), "Benefits | Adobe Flash runtimes," https://www.adobe.com/products/flashruntimes/benefits.html

Babylon.js (2018), "BabylonJS – 3D Engine based on WebGL/Web Audio and Java Script," https://www.babylonjs.com/

Balci, O. (1988), "The Implementation of Four Conceptual Frameworks for Simulation Modeling in High-level Languages," In *Proceedings of the 1988 Winter Simulation Conference* (San Diego, CA, Dec. 12-14). IEEE, pp. 287-295.

Boukerche, A., Iwasaki, F. M., Regina, A. B., and Pizzolato, E. B. (2008), "Web-Based Distributed Simulations Visualization and Control with HLA and Web Services," In *12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications* (Vancouver, BC, Canada, Oct. 27-28). IEEE, pp. 17-23.

Byrne, J., Heavey, C. and Byrne, P.J. (2010), "A review of Web-based simulation and supporting tools," *Simulation Modelling Practice and Theory 18*, 3, 253-276.

De Lara, J., and Alfonseca, M. (2001), "Constructing Simulation-Based Web Documents," *IEEE MultiMedia 8*, 1, 42-49.

Fishwick, P. (1996). "Web-based simulation: some personal observations," In *Proceedings of the 1996 Winter Simulation Conference* (Coronado, CA, Dec. 8-11). IEEE, pp. 772-779.

Gan, B. P., Liu, L., Ji, Z., Turner, S. J., and Cai, W. (2001), "Managing Event Traces For a Web Front-End to a Parallel Simulation," In *Proceedings of the 2001 Winter Simulation Conference* (Arlington, VA, Dec. 9-12). IEEE, pp. 637–644.

Gocmenoglu, C., and Acarman, T. (2014), "A 3D Web-based Visualization Tool for VANET Simulations," In *2014 International Conference on Connected Vehicles and Expo* (Vienna, Austria, Nov. 3-7). IEEE. pp. 827-828.

Google (2018), "Google Maps," https://www.google.com/maps

Graupner, T.-D., Richter, H., and Sihn, W. (2002), "Configuration, Simulation and Animation of Manufacturing Systems Via the Internet," In *Proceedings of the 2002 Winter Simulation Conference* (San Diego, CA, Dec. 8-11). IEEE, pp. 825-829.

Henriksen, J. O., Lorenz, P., Hanisch, A., Osterburg, S., and Schriber, T. J. (2002), "Web Based Simulation Center: Professional Support for Simulation Projects," In *Proceedings of the 2002 Simulation Conference* (San Diego, CA, Dec. 8-11). IEEE, pp. 807–815.

IEEE (2000), "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems," IEEE STD 1471-2000, New York, NY.

Jeong, D., Seo, M., and Seo, Y. (2009), "Development of Web-Based Simulator for Supply Chain Management," In *Proceedings of the 2009 Winter Simulation Confrence* (Austin, TX, Dec. 13-16). IEEE. pp. 2303-2309.

Kim, T., Lee, J., & Fishwick, P. (2002), "A Two-Stage Modeling and Simulation Process for Web-Based Modeling and Simulation," *ACM Transactions on Modeling and Computer Simulation 12*, 3, 230-248.

Lorenz, P., Dorwarth, H., Ritter, K.-C., and Schriber, T. (1997), "Towards A Web Based Simulation Environment," In *Proceedings of the 1997 Winter Simulation Conference* (Atlanta, GA, Dec. 7-10). IEEE, pp. 1338-1344.

Microsoft (2018), "Download DirectX End-User Runtime Web Installer," https://www.microsoft.com/en-us/download/details.aspx?id=35

Mwalongo, F., Krone, M., Becher, M., Reina, G., and Ertl, T. (2015), "Remote Visualization of Dynamic Molecular Data using WebGL," In *Proceedings of the 20th International Conference on 3D Web Technology* (Heraklion, Crete, Greece, Jun. 18-21). ACM. pp. 115-122.

Myers, D. (2004), "An Extensible Component-Based Architecture for Web-Based Simulation Using Standards-Based Web Browsers," M.S. Thesis, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA.

Myers, D. S., and Balci, O. (2009), "A Web-Based Visual Simulation Architecture," *International Journal of Modelling and Simulation, 29*, 2, 137-148.

Oracle (2018), "The Client Tier," https://docs.oracle.com/cd/E19226-01/820-7759/gcrla/index.html

Park, M. (2002), "SIMPACKJ/S: A Web-oriented Toolkit for Discrete Event Simulation," M.S. Thesis, Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL.

Padilla, J. J., Diallo, S. Y., Barraco, A., Lynch, C. J., and Kavak, H. (2014), "Cloud-Based Simulators: Making Simulations Accessible to Non-Experts and Experts Alike," In *Proceedings of the 2014 Winter Simulation Conference* (Savanah, GA, Dec. 7-10). IEEE. pp. 3630-3638.

Salisbury, C. F., Farr, S. D., and Moore, J. A. (1999), "Web-Based Simulation Visualization Using Java3D," In *Proceedings of the 1999 Winter Simulation Conference* (Phoenix, AZ, Dec. 5-8). IEEE, pp. 1425-1429.

Sawicki, B., and Chaber, B. (2013), "Efficient visualization of 3D models by web browser," *Computing, 95*, 661-673.

Schutt, K., and Balci, O. (2016), "Cloud Software Development Platforms: A Comparative Overview," In *14th International Conference on Software Engineering Research, Management and Applications* (Towson, MD, Jun. 8-10). IEEE.

Three.js (2018), "three.js – JavaScript 3D Library," https://threejs.org/

Wang, Y.-H., and Liao, Y.-C. (2003), "Implementation of a Collabrative Web-based Simulation Modeling Environment," In *Proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications* (Delft, Netherlands, Oct. 23-25). IEEE.

Web3D (2018), "Web3D Consortium," http://www.web3d.org/

X3Dom (2018), "x3Dom," https://www.x3dom.org/