Proceedings of the 2019 Winter Simulation Conference
N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, eds.

ARCHITECTURE AND DESIGN OF A CLOUD-BASED VISUAL SIMULATION ENVIRONMENT

Sait Tuna Önder
Osman Balci

Department of Computer Science
Virginia Polytechnic Institute and State University (Virginia Tech)
Blacksburg, Virginia 24061, USA

ABSTRACT

Doing things "in the cloud" has become ubiquitous and the "cloud" has become a rich platform for the use of modeling and simulation (M&S) anywhere and anytime to provide solutions to complex problems. Creation of an Integrated Development Environment (IDE) for building and executing visual M&S applications "in the cloud" poses significant technical challenges. This paper presents an architecture and a design of a cloud-based visual M&S IDE for the example problem domain "traffic networks". The IDE consists of integrated software tools that provide computer-aided assistance in the composition and visualization of simulation models under a web browser on a client computer while the simulation model is being executed on a server computer. Based on a client-server architecture enabling distributed multitiered M&S development, the design employs an asynchronous visualization protocol with efficient resource utilization. The architecture and design can assist researchers and developers to create other cloud-based visual M&S IDEs.

1 INTRODUCTION

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 paper addresses the technical challenge of how to create and execute a visual modeling and simulation (M&S) application "in the cloud".

We developed a distributed multitiered software system as a research prototype called cloud-based visual simulation environment for traffic networks (CANVAS). We selected traffic networks as an example problem domain to illustrate the architecture and design of a visual M&S Integrated Development Environment (IDE) in CANVAS. Using a web browser on an Internet-connected computer, the user can develop and execute a visual simulation of a traffic network "in the cloud" with the tools made available in CANVAS research prototype, currently available at (CANVAS 2019).

The objective of this paper is to present an architecture and a design of a cloud-based visual M&S IDE to assist researchers and developers to create other cloud-based visual M&S IDEs.

The remainder of this paper is organized as follows. Section 2 introduces the architecture. Section 3 describes the design. A case study is presented in Section 4. Section 5 states concluding remarks.

2 ARCHITECTURE OF A CLOUD-BASED VISUAL M&S IDE

"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 2000) We created the CANVAS architecture on top of the Java platform, Enterprise Edition (Java EE) client-server architecture, which consists of the five tiers as depicted in Figure 1.

Java EE (Oracle 2019) is a specification based on which many companies have created commercial products for implementing its component technologies. This enables maintainability and reusability in development. A Java EE component technology product can easily be replaced with another as long as the product conforms to the Java EE specification for the version desired. For example, many Java EE application server products are available such as IBM WebSphere, GlassFish, and WildFly. Many Relational Database Management System (RDBMS) products are available such as DB2, MySQL, and Oracle. Each can be replaced with another very easily since each conforms to the Java EE specification.

Commonly used Model View Controller (MVC) architectural pattern is embedded within the five tiers shown in Figure 1: *View* corresponds to the client tier, *Controller* corresponds to the web tier, and *Model* corresponds to the business tier, data mapping tier, and data source tier.

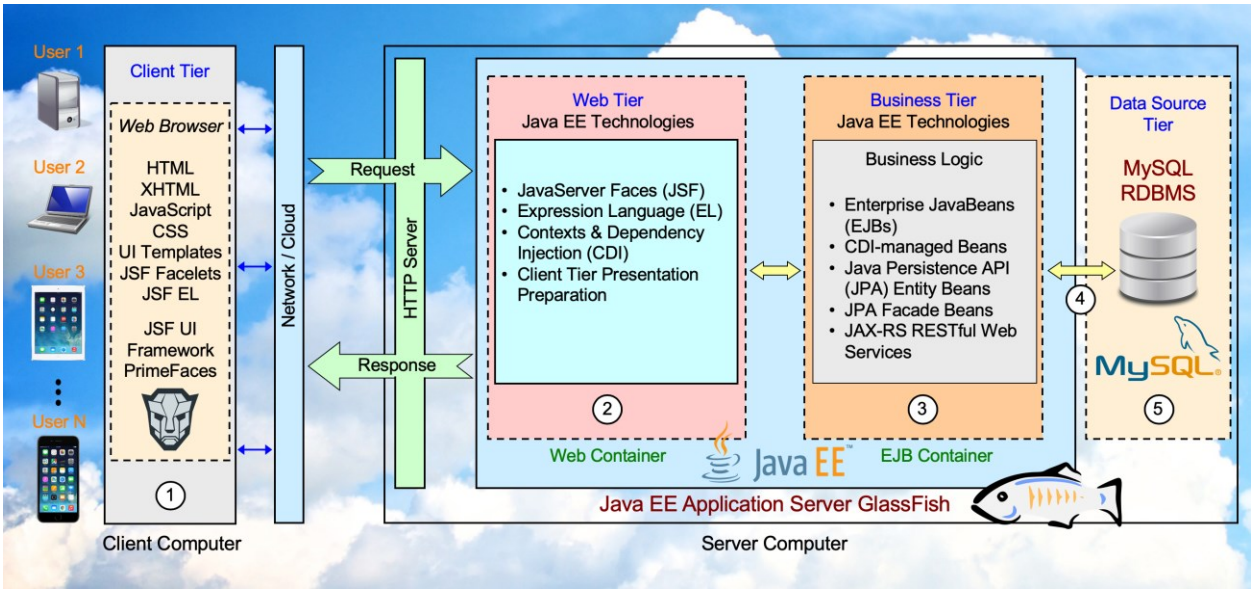

Figure 1: Java EE Client-Server Architecture of CANVAS.

2.1 Client Tier

The client tier provides the user interface of the CANVAS IDE in a web browser such as Google Chrome, Safari, Firefox, or Microsoft Edge. The CANVAS IDE provides tools for creating and running a visual simulation model of an example problem domain selected as "traffic networks". Visualization of the simulation model running on the server computer is accomplished by using Three.js, which is a JavaScript library and Application Programming Interface (API) based on WebGL for creating and displaying animated 3D computer graphics in a web browser (Three.js 2019).

2.2 Web Tier

The web tier controls the interaction between the user in the client tier and the simulation model running in the business tier by using Java EE technologies such as JavaServer Faces (JSF), Expression Language (EL), and Context & Dependency Injection (CDI).

The communication between the simulation model visualization in the client tier and the simulation model execution in the business tier must be as fast as possible. Going over the HTTP server would be too slow. Therefore, we used WebSockets for this communication in CANVAS. WebSocket (WebSocket 2019) is a computer communications protocol, which provides full-duplex bi-directional communication channels over a single Transmission Control Protocol (TCP) connection enabling much better performance than communicating over the HTTP server.

2.3 Business Tier

Simulation model is executed in the business tier, which provides Java EE technologies such as Enterprise JavaBeans (EJBs), CDI-managed beans, Java Persistence API (JPA) entity beans, JPA facade beans, and Java API for RESTful Web Services (JAX-RS). The simulation model is created under a conceptual framework in which traffic network entities such as vehicles, move spots, traffic lights, forks, and merges are represented as static and dynamic objects. The server-side components of the IDE such as simulation builder, simulation manager, and message manager are executed in the business tier.

The simulation model can call RESTful web services running on other server computers to perform some of its work. For example, the random variate generation web services created by Sabah and Balci (2005) can be called to obtain random variates needed during model execution. Utilizing such external web services in the business tier forms the Service Oriented Architecture (SOA) in the backend resulting in an overall architecture which becomes a combination of client-server architecture and SOA.

2.4 Data Mapping Tier

The data mapping tier establishes the connection between the business tier and the data source tier using Java EE technologies such as Java Database Connectivity (JDBC) API, Java Persistence API (JPA), Java EE Connector Architecture, and Java Transaction API (JTA).

Storage and retrieval of simulation models, user's data, and M&S project data are carried out using a RDBMS. Under the relational database paradigm, data is represented in tables and manipulated using the Structured Query Language (SQL). However, we build a simulation model under the Object Oriented paradigm in which we create classes, from which we instantiate objects that communicate with each other via message passing. Hence, a mismatch occurs between the two paradigms.

JPA is created to resolve this mismatch. It enables the representation of a database table as an entity class where inserting a new row in a database table corresponds to instantiating an object from the entity class representing the table. The instantiated object holds the data contained in a database table row. JPA maps the tabular database paradigm to the object oriented paradigm.

2.5 Data Source Tier

The data source tier consists of a MySQL RDBMS in CANVAS for storage and retrieval of simulation models, user's data, and M&S project data. Many RDBMS products are available to use in this tier, which conform to the Java EE specification, including DB2, MySQL, Oracle, and PostgreSQL.

3 DESIGN OF A CLOUD-BASED VISUAL M&S IDE

A design is an instantiation from an architecture similar to how an object is an instantiation from a class. Therefore, CANVAS design is created as an instance from the architecture described in Section 3.

3.1 CANVAS IDE Toolset

CANVAS IDE is an integrated set of software tools that provide computer-aided assistance throughout the M&S development life cycle for the example problem domain "traffic networks". Its toolset consists of Model Composer, Simulation Builder, Simulation Manager, Message Manager, and Visualization Manager as depicted in Figure 2.

3.1.1 Model Composer

Problem domain-specific modeling enables component-based model development by way of reuse. Therefore, we selected "traffic networks" as the example problem domain. Entities of interest (e.g., traffic light, vehicle, enter point, exit point, move spot, merge, fork) in the problem domain (e.g., traffic networks) can be characterized as components. A reusable simulation model of each component can be built and

provided as a reusable object within the IDE. The modeler is enabled to compose a model by way of reusing the objects representing entities of interest in the problem domain as depicted in Figure 3.

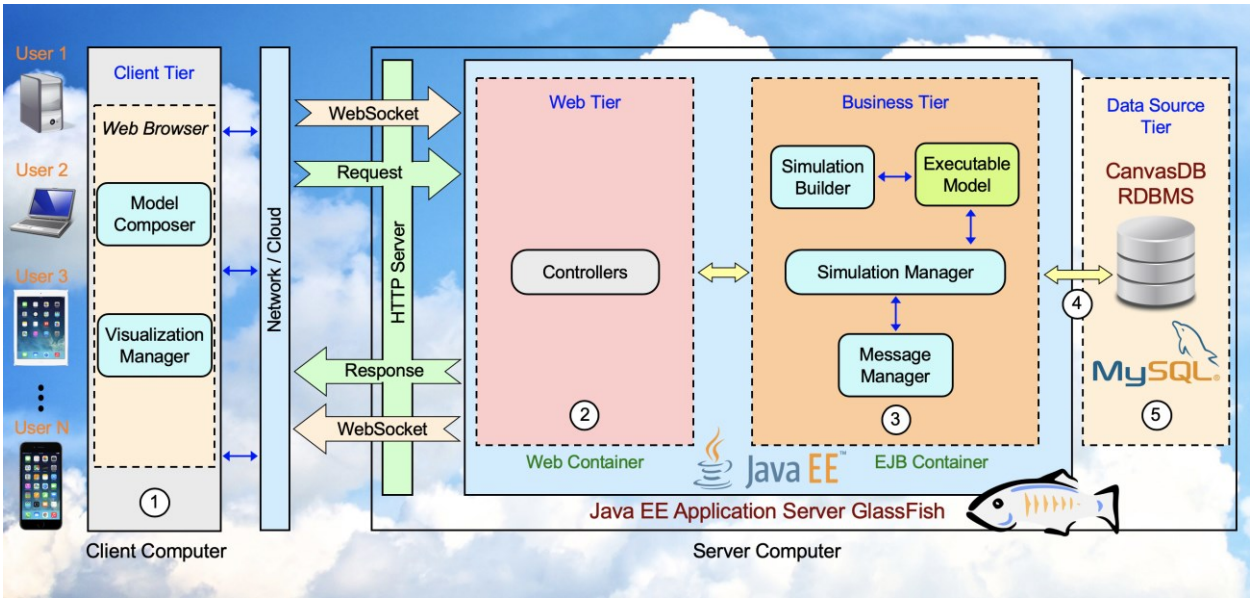

Figure 2: CANVAS Design.

CANVAS requires the upload of a top-down satellite image of a traffic network to simulate as depicted in Figure 3. The modeler reuses the traffic light object by placing it on the satellite image and specifying its attributes. Similarly, a vehicle's enter and exit points, move spots, forks, and merges are represented by way of reuse. Interarrival times of vehicles to a particular lane in an enter point are also specified. Vehicle sizes can be adjusted based on the size of the uploaded traffic network image.

3.1.2 Simulation Builder

Model validation plays a crucial role in simulation model development. Our CANVAS IDE is created for non-experts and 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 credibility of the simulation model. Therefore, a *Model Validator* must provide 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 a component of the *Simulation Builder* under the CANVAS design specification. A simulation 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 model validity checks are performed at the same time. This operation produces a graph representing a traffic network. The *Model Validator* ensures 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.

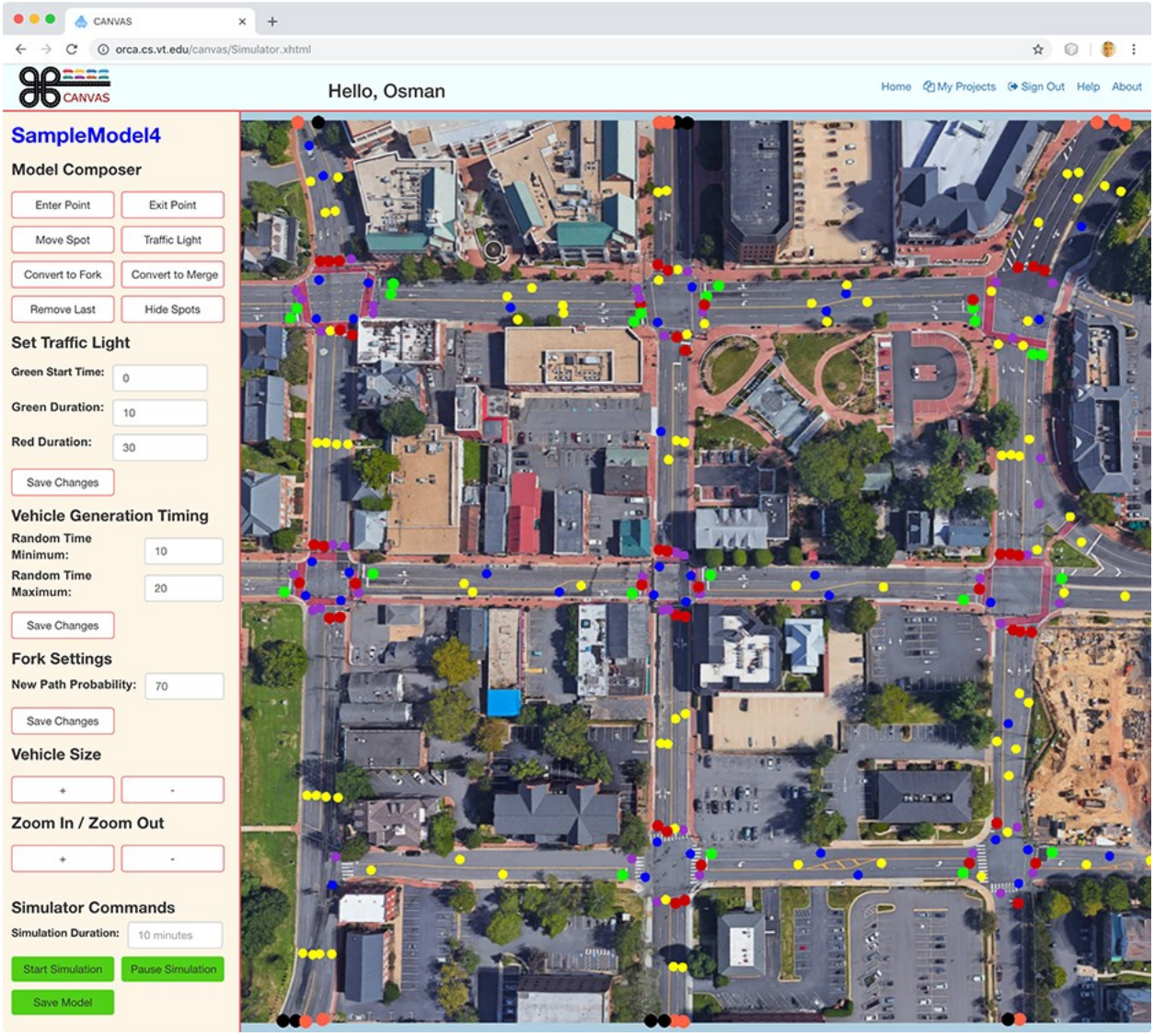

Figure 3: Composition of a Visual Simulation Model in CANVAS.

3.1.3 Simulation Manager

The *Simulation Manager* is the core of the CANVAS design specification. The traffic network simulation algorithms run inside this tool with the input received from the *Simulation Builder*. It continuously produces visualization data and delivers them to the *Message Manager*. All events that can be visualized are produced by this tool.

A thread-safe Simulation Manager instance is created for each user session. Each Simulation Manager instance enters a waiting stage while the required number of events are generated. During the waiting period, system resources can be allocated for other tasks. When a user client consumes events, execution continues to feed the user client with more events. In case of user client pausing the visualization, the simulation manager automatically enters a waiting stage since the client stops requesting new messages.

3.1.4 Message Manager

The *Message Manager* has a buffer that stores messages produced by the *Simulation Manager* before sending them to the client. The messages are formatted in JavaScript Object Notation (JSON) and are stored in the buffer until the client requests them. Since the client is in control of the message data flow, it cannot be overwhelmed by messages sent from the server.

The messages are ordered in the buffer with respect to event occurrence times before sending them to the client. The buffer utilizes a priority queue data structure for message ordering. The Simulation Manager does not necessarily produce messages in real time order. For example, at current time t, an enter point object can schedule a vehicle arrival event for time t+10. After that, 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 arrival event even though it is generated later.

3.1.5 Visualization Manager

The *Visualization Manager* creates the visual simulation under the web browser as depicted in Figure 4 based on the messages produced by the *Simulation Manager* and sent by the *Message Manager*. Since static objects do not move, the main responsibility of the visualization manager is to update coordinates of dynamic objects. Each dynamic object possesses speed and rotation attributes that are used to move the object every time increment.

The *Visualization Manager* buffers the events with respect to increasing order of event occurrence times. 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 increment, the *Visualization Manager* only updates the coordinates of dynamic objects according to their speed and direction. The *Visualization Manager* can process different types 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 an image to an 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, where yellow is included within the duration time of green.

3.2 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 by the user on the client side under a web browser. The simulation model 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 object generates a "vehicle creation" event or a traffic light object 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 by the *Message Manager* with their timestamp 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 increment.

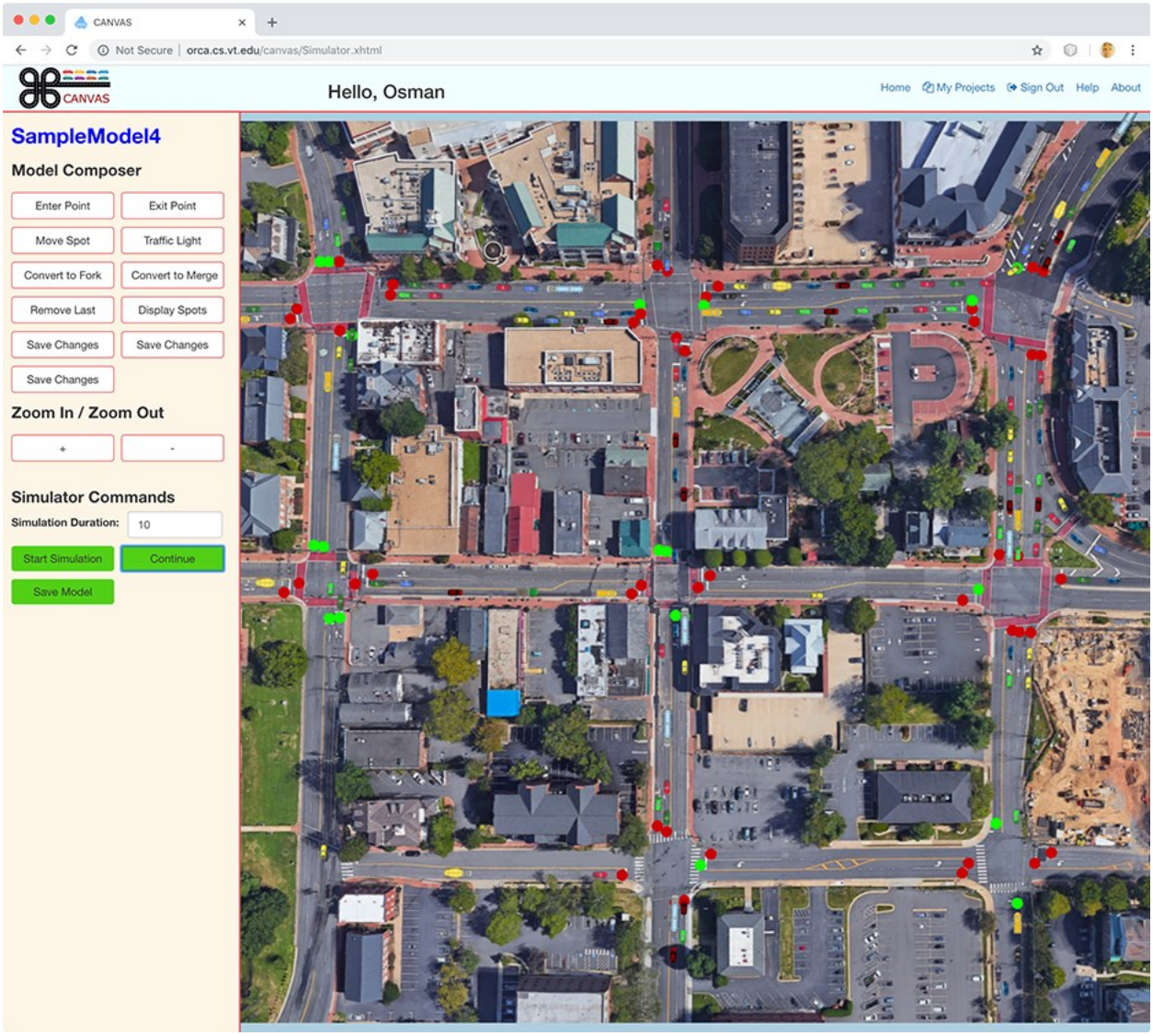

Figure 4: Visual Simulator in CANVAS.

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 hundred messages in every two minutes. However, the client can be easily overloaded 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 overloaded.

Users watch the simulation visualization in 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 not to execute the 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 memory 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 set of 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 with the execution to fill the buffer.

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 Figure 5 and each circled number is described below.

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 event with the earliest event occurrence time.
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 the new messages into its buffer, which is a priority queue, with respect to the timestamps.

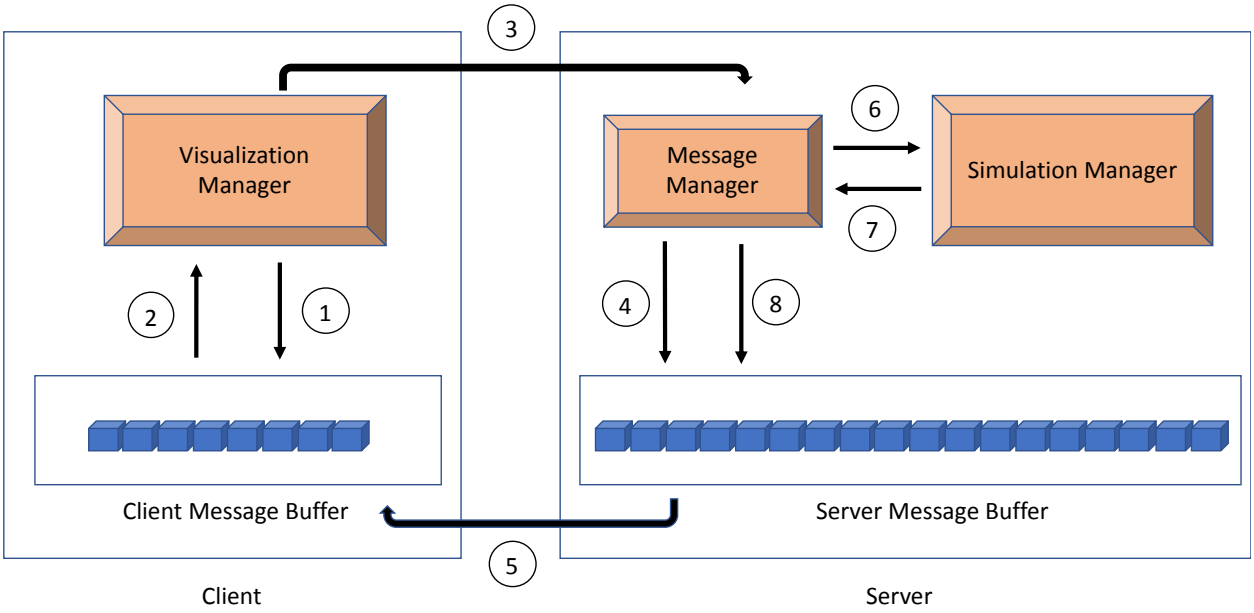

Figure 5: CANVAS Event Request.

4 A CASE STUDY

CANVAS can visually simulate complex traffic networks that consist of multiple traffic intersections. A user creates an account and signs in to CANVAS (CANVAS 2019). In creating a new simulation model, the user obtains the satellite map view of a traffic network using a map app such as Google Maps. The satellite map image is uploaded into CANVAS to be the background image upon which to compose the visual simulation model as depicted in Figure 6. The background map image is placed at the center of the visualization area within the web browser. Refreshing the web page after changing the web browser window size automatically resizes the background map image.

In this case study, we provide a visual simulation model of a traffic network in Fairfax, Virginia, USA. Figure 6 shows the composed model that has 19 enter points and 267 static objects in total.

Each intersection is carefully designed to provide a realistic representation of the traffic network. Model components that have different colors have their own unique behavior. Red and green spots represent traffic lights and each traffic light is assigned a green light duration and a red light duration. Yellow light duration is include within the green light duration since most drivers pass on yellow.

Enter points are represented with orange color. They are assigned a lower limit L and upper limit U. The vehicles are generated randomly to enter the intersection based on an interarrival time random variable with uniform probability distribution between L and U.

Black color represents exit points where vehicles depart the intersection and are removed from the simulation. Forks are represented with blue color where vehicles can change their direction between two alternative paths. When vehicles arrive at a fork point, they choose a direction according to direction probability that is assigned to the point. Purple points represent merges where two paths merge into one. Standard move spots are represented with yellow color.

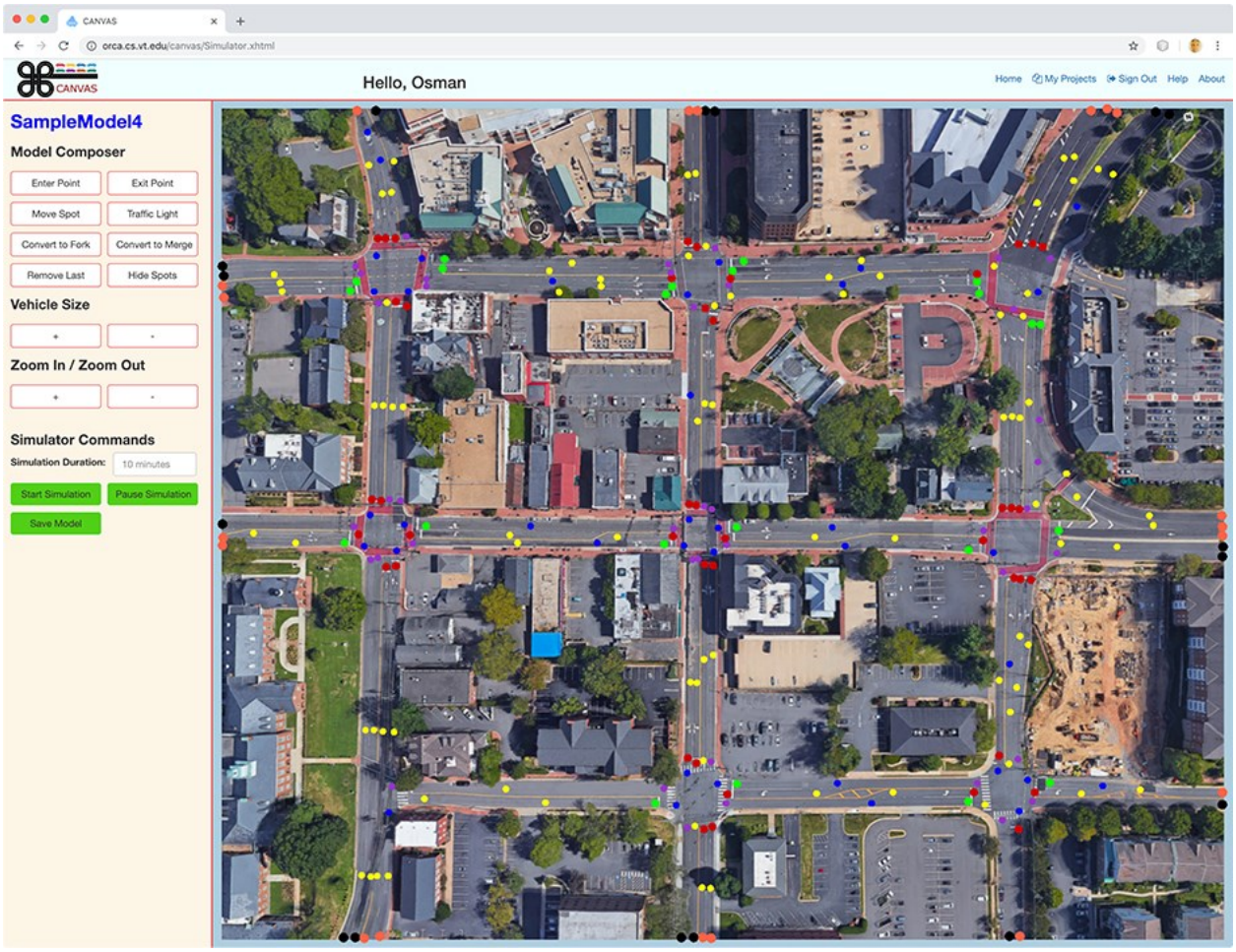

Figure 6. Case Study Visual Simulation Model Composition.

Figure 7 shows the visual simulation execution in this case study. We set the simulation duration to 30 minutes and click *Start Simulation* to start the simulation model execution. Vehicle generation is terminated after 30 minutes while the model execution continues until the last vehicle leaves the simulation model.

The simulation execution generated 2938 vehicles that spent 470 seconds on average in the traffic network. Visualizing a complex simulation model for a long time requires high number of events. According to server logs, 258968 events were generated to provide the visualization in this case study. These events include vehicle generation and destructions as well as speed and direction changes of vehicles.

5 CONCLUDING REMARKS

This paper demonstrates the technical feasibility of creating a Visual Simulation Environment in the form of an Integrated Development Environment (IDE) for composing and executing visual M&S applications "in the cloud" under a commonly used web browser. The problem domain "traffic networks" is selected so that visual simulation model development is facilitated with component-based model composition. Any other problem domain could have been chosen. Therefore, the purpose of the work presented herein is not about visual simulation of traffic networks; it is about demonstrating how an IDE can be created to provide integrated software tools that provide computer-aided assistance in the composition and visualization of simulation models under a web browser on a client computer while the simulation model is being executed on a server computer.

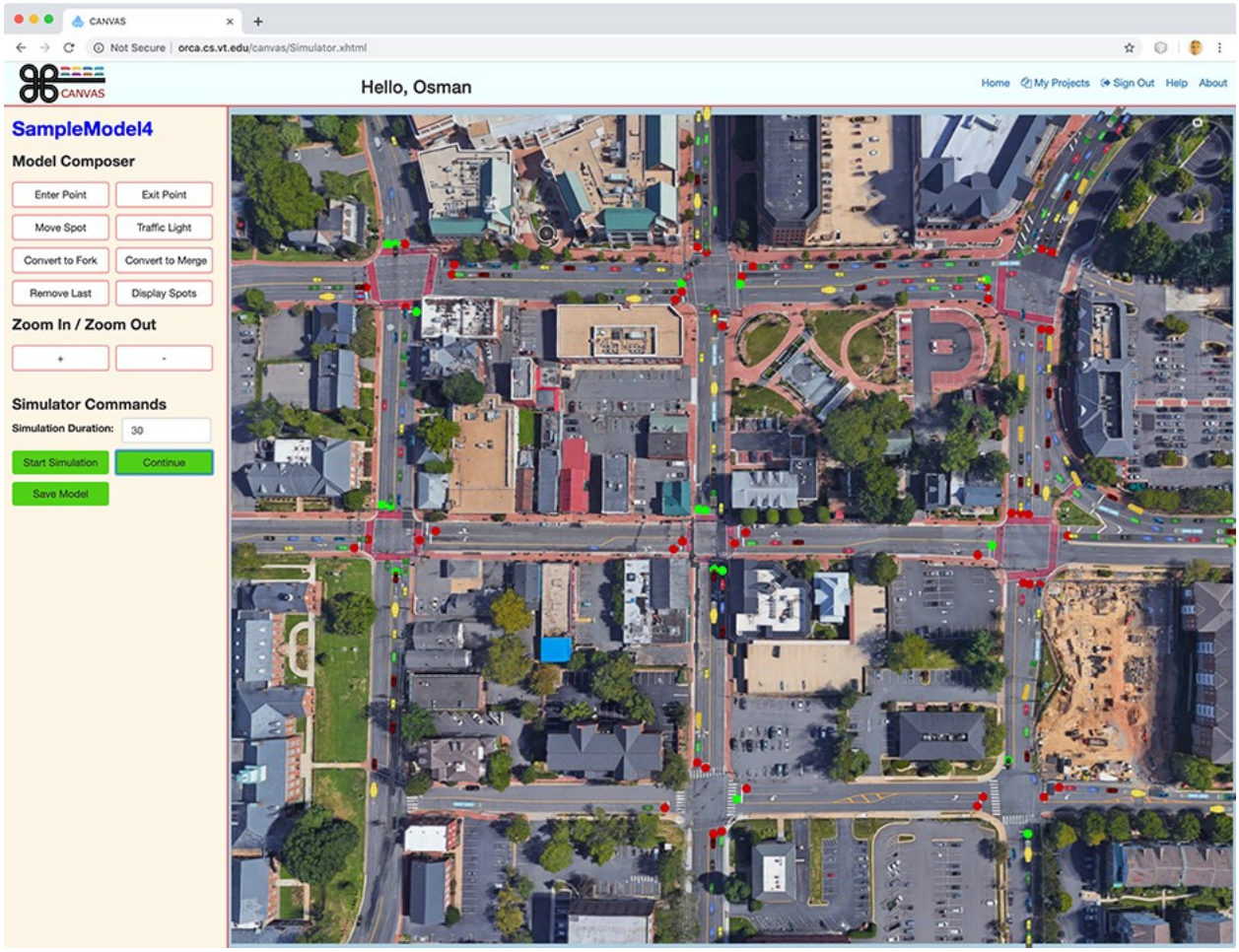

Figure 7: Case Study Visual Simulation Model Visualization.

This paper presents the architecture and design of the cloud-based visual M&S IDE under the Java platform, Enterprise Edition (Java EE) using GlassFish application server and MySQL relational database management system. Other software products conforming to the Java EE specification could have been used instead of GlassFish and MySQL.

A cloud-based visual M&S IDE can also be designed and developed under platforms other than Java EE. For example, the following platforms or frameworks can be used: Microsoft platform .NET framework (Microsoft 2019), Ruby on Rails framework (Ruby on Rails 2019), Zend framework (Zend 2019), Node.js framework (Node.js 2019), or Django (Python) framework (Django 2019). Schutt and Balci (2016) present a comparative overview of these platforms / frameworks.

REFERENCES

CANVAS. 2019. "CANVAS website," http://orca.cs.vt.edu/canvas/, accessed 29th July 2019.

Django. 2019. "Django Software Foundation," https://www.djangoproject.com/, accessed 29th July 2019.

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

Microsoft. 2019. "Microsoft .NET Framework," Microsoft Corporation, Redmond, WA, https://dotnet.microsoft.com/, accessed 29th July 2019.

Node.js. 2019. "Node.js Foundation," https://nodejs.org/, accessed 29th July 2019.

Oracle. 2019. "Java EE Overview," https://www.oracle.com/technetwork/java/javaee/overview/, accessed 29th July 2019.

Ruby on Rails. 2019. "Ruby on Rails," https://rubyonrails.org/, accessed 29th July 2019.
Sabah, M. and O. Balci. 2005. "Web-based Random Variate Generation for Stochastic Simulations," *International Journal of Simulation and Process Modelling* 1(1-2):16-25.
Schutt, K. and O. Balci. 2016. "Cloud Software Development Platforms: A Comparative Overview," In *Proceedings of the IEEE 14th International Conference on Software Engineering Research, Management and Applications* (SERA), Baltimore, MD, June 8-10, 3-13. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
Three.js. 2019. "three.js," https://en.wikipedia.org/wiki/Three.js, accessed 29th July 2019.
WebSocket. 2019. "WebSocket," https://en.wikipedia.org/wiki/WebSocket, accessed 29th July 2019.
Zend. 2019. "Zend Framework," https://framework.zend.com/, accessed 29th July 2019.

AUTHOR BIOGRAPHIES

SAIT TUNA ÖNDER is a Software Engineer at Bloomberg L.P. He obtained his B.S. degree in Computer Engineering from Bilkent University (Ankara) in 2014 and his M.S. degree in Computer Science from Virginia Polytechnic Institute and State University (Virginia Tech) in 2018. His areas of expertise focus on Modeling and Simulation and Cloud Software Development. His email address is s.tunaonder@gmail.com.

OSMAN BALCI is Professor of Computer Science and Director of the Mobile/Cloud Software Engineering Lab at Virginia Polytechnic Institute and State University (Virginia Tech). He received his B.S. and M.S. degrees from Boğaziçi University (Istanbul) in 1975 and 1977, and M.S. and Ph.D. degrees from Syracuse University (New York) in 1978 and 1981. Dr. Balci currently serves as Editor-in-Chief of *ACM SIGSIM Modeling and Simulation (M&S) Knowledge Repository* and editorial board member of several journals. He served as the Founding Editor-in-Chief of two international journals: *Annals of Software Engineering*, 1993-2002 and *World Wide Web*, 1996-2000. He served as Chair of ACM SIGSIM, 2008-2010; and Director at Large of the Society for Modeling and Simulation International (SCS), 2002-2006. Dr. Balci received the Virginia Tech 2018 W.S. 'Pete' White Innovation in Engineering Education Award, the 2015 ACM SIGSIM Distinguished Contributions Award, and the 2013 McLeod Founder's Award for Distinguished Service to the Profession given by SCS. His current areas of expertise center on iOS (iPhone, iPad, iPod touch) Mobile Software Engineering, Cloud Software Engineering, architecting cloud software-based system of systems, software/system IV&V; Modeling and Simulation (methodology, verification, validation, and certification); and Digital Game-Based Learning. His e-mail and web addresses are balci@vt.edu and https://manta.cs.vt.edu/balci.