A CONNECTED-VEHICLE TRAFFIC SIGNAL SYSTEM MODELING PLATFORM

FINAL PROJECT REPORT

by

Robert B. Heckendorn, Neeta A Eapen, and Ahmed Abdel-Rahim
University of Idaho

Sponsorship
PacTrans and the University of Idaho

for
Pacific Northwest Transportation Consortium (PacTrans)
USDOT University Transportation Center for Federal Region 10
University of Washington
More Hall 112, Box 352700
Seattle, WA 98195-2700

In cooperation with US Department of Transportation-Research and Innovative Technology Administration (RITA)

Disclaimer

Technical Report Documentation Page

1. Report No.	**2. Government Accession No.** 01723933	**3. Recipient's Catalog No.**
4. Title and Subtitle A CONNECTED-VEHICLE TRAFFIC SIGNAL SYSTEM MODELING PLATFORM		**5. Report Date**
		6. Performing Organization Code
7. Author(s) Robert B. Heckendorn, 0000-0001-9756-554X; Neeta A Eapen, and Ahmed Abdel-Rahim, 0000-0001-9756-554X		**8. Performing Organization Report No.** 2019-S-UI-3
9. Performing Organization Name and Address PacTrans Pacific Northwest Transportation Consortium University Transportation Center for Region 10 University of Washington More Hall 112 Seattle, WA 98195-2700		**10. Work Unit No. (TRAIS)**
		11. Contract or Grant No. 69A3551747110
12. Sponsoring Organization Name and Address United States of America Department of Transportation Research and Innovative Technology Administration		**13. Type of Report and Period Covered**
		14. Sponsoring Agency Code

15. Supplementary Notes
Report uploaded at www.pacTrans.org

16. Abstract

Traffic in the cities of the future will be managed by sophisticated signaling systems working in conjunction with network-connected vehicles that share a wide variety of data. A common configuration for information flow will be from vehicles on the road to roadside units (RSUs) via basic safety messages (BSM). A BSM contains information about position, heading, state of the vehicle, etc. The roadside infrastructure (RSI) may in return convey information about traffic conditions ahead such as the status of signaling. Traffic control algorithms can then plan and improve traffic control by managing signals via protocols such as signal phasing and timing (SPaT) messages. Traffic simulators are critical tools to aid traffic planners in modeling traffic signal operations and traffic flow. VISSIM is a popular example of this class of tools. Unfortunately, VISSIM does not support BSM and SPaT protocols, thus limiting the development of simulations of the more closely connected and managed traffic environments of the future. This software project enables modeling of traffic systems of the future by suppling a well-documented and technically precise implementation of BSM and SPaT programming interfaces for VISSIM.

17. Key Words Traffic simulation, connected vehicles, BSM, SPaT, VISSIM		**18. Distribution Statement** No restrictions.	
19. Security Classification (of this report) Unclassified.	**20. Security Classification (of this page)** Unclassified.	**21. No. of Pages**	**22. Price** NA

Form DOT F 1700.7 (8-72) **Reproduction of completed page authorized**

Table of Contents

List of Abbreviations viii

Executive Summary ix

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 MODEL DEVELOPMENT 3

2.1 Statement of Purpose 3

2.2 Model Overview 4

2.3 Preliminaries 5

2.4 Object Creation for Using CoVeTTware 5

2.5 Function for Safe Closing 6

2.5.1 Public void runVissimOnce() 6

2.5.2 Public int numberOfRunsFromSecs(double t) 6

2.6 Basic Safety Message (BSM) 7

2.6.1 BSM Fields 7

2.6.2 Functions for BSM 8

CHAPTER 3 MODEL IMPLEMENTATION AND TESTING 13

3.1 Model Implementation Examples 13

3.1.1 Display a BSM Message 13

3.1.2 Report Movement in the Network 13

3.1.3 Report Movement of an Intersection in the Network, When Intersection Number Is Given 14

3.1.4 Report Movement of an Intersection in the Network, When Intersection Name Is Given 15

3.1.5 Report Movement of the Signal Head of an Intersection in the Network, When the Signal Head Number Is Given 16

3.1.6 Report Movement of the Signal Head of an Intersection in the Network, When the Signal Head Name Is Given 16

3.2 Signal Phasing and Timing Message (SPaT) 17

3.2.1 SPaT Fields 17

3.3 Functions for SPaT 19

3.3.1 Public spatSignalGroupData createSPATSignalGroupData(int signalGroupId, String phaseState, float minEnd- Time) 19

3.3.2 Public SPAT createSPAT(int intersectionNumber, String intersecStatus, List<spatSignalGroupData> sgData) 19

3.3.3 Public SPAT createSPAT(String signalControllerName, String intersecStatus, List<spatSignalGroupData> sg- Data) 20

3.3.4 Public void setSPAT(SPAT s) 20

3.3.5 Public void displaySPAT(SPAT s) 20

3.3.6 Public void writeSPAT(SPAT s) 21

3.4 SPaT Implementation Example 21

3.4.1 Control Signals of an Intersection in the Network, When Intersection Number Is Given 21

3.4.2 Control Signals of an Intersection in the Network, When Intersection Name Is Given 22

3.5 Testing of a Connected Vehicle Traffic Control Algorithm 23

3.5.1 An Example of a Fixed-Time Signal Generation Algorithm 24

CHAPTER 4 CONCLUSIONS AND RECOMMENDATIONS 27

CHAPTER 5 REFERENCES 29

List of Figures

Figure 2-1 Data Exchange among CoVeTTware, VISSIM, and the Traffic Control Algorithm .. 4

Figure 2-2 Register COM Server Command in VISSIM.. 5

List of Tables

Table 2-1 BSM Fields 8

Table 3-1 SPaT Fields 19

Table 3-2 SPaT Signal Group Data fields 19

List of Abbreviations

API	Application programming interface
BSM	Basic safety message
COM	Component object model
CoVeTTware	Connected Vehicle Traffic control algorithm Testing Software
IEEE	Institute of Electrical and Electronics Engineers
RSI	Roadside infrastructure
RSU	Roadside unit
SAE	Society of Automotive Engineers
SPaT	Signal phasing and timing

Executive Summary

VISSIM is a popular micro-simulation package for traffic analysis. By micro-simulation we mean that it simulates the behavior of individual cars in motion in traffic and breaks up time into small steps. Unfortunately, VISSIM lacks an application programming interface (API) for connected vehicles and roadside infrastructure development. In response, this was a technology enabling project rather than a research project. Software was designed and implemented to augment enable the VISSIM simulator to model a connected-vehicle environment. This included interfaces that allow the design and creation or roadside infrastructure (RSI) software that accesses realistic and precise basic safety messages (BSM) from VISSIM and the controlling of signals in VISSIM via signal phasing and timing (SPaT) commands, as if it were connected to real traffic hardware and controlling real traffic.

This project aimed to design and develop Connected Vehicle Traffic control algorithm Testing Software (CoVeTTware). The traffic simulation in this work was done by using the VISSIM simulation tool. The component object model (COM) interface was used to access the objects of the VISSIM simulation tool. The exchange of information between VISSIM and CoVeTTware was performed in real time. CoVeTTware retrieves information about vehicles at intersections from VISSIM and generates the BSMs of the vehicles in real time, according to the Society of Automotive Engineers (SAE) J2735 standard. The BSMs can be used by a connected vehicle traffic control algorithm to generate signal information for an intersection. The signal information includes the signal phase to be set for a signal and the period for which the signal phase should be set. The signal information is used by CoVeTTware to generate SPaT messages, according to the SAE J2735 standard. Using CoVeTTware, the statuses of the signals of the

intersection are changed in VISSIM in real time by on the basis of the SPaT messages. When the period of the given signal phase expires, CoVeTTware resets the signal phase as UNDEFINED.

This model can be used to test any connected vehicle traffic control algorithm. The software has been tested, and a manual is supplied.

CHAPTER 1 INTRODUCTION

Traffic in the cities of the future will be managed by sophisticated signaling systems working in conjunction with networks of connected vehicles that share a wide variety of data. A common configuration for information flow is from vehicles on the road to roadside units (RSUs) via basic safety messages (BSM), (American Society for Testing and Materials (ASTM) 2010). A BSM contains information about the position, heading, and state of the vehicle. The roadside infrastructure (RSI) may in return convey information about traffic conditions ahead such as the status of signaling. These systems may allow drivers to better plan by having richer knowledge of the traffic conditions and transportation infrastructure in which they are embedded. The roadside infrastructure can similarly help improve traffic flow and safety.

Traffic planners often use simulations to predict traffic behavior. A micro-simulation uses traffic objects such as cars, buses, and pedestrians and moves them through a simulated world a small distance at a time. This enables careful simulation of interactions between traffic objects. VISSIM is a popular micro-simulation package and was the focus of our work.

This project was a technology and research enabling project more than strictly a research project. We carefully created an application programming interface (API) for VISSIM based on standards that conform BSM messages from cars arriving at intersections, as if RSUs were in place to monitor traffic (Heckendorn and Eapen 2021). The design is extensible to any placement of RSUs. An API was also created to accept the same standardized signal phase and timing (SPaT) messages that real RSI would accept for signal control. With these APIs based on real standards for these message protocols, software engineers can construct systems that simulate traffic monitoring, signal control, and decision platforms such as traffic control and optimization. Because the APIs simulate real messages, the input and output from the traffic control software can easily be switched from connections to the simulator to connections to real standards-compliant hardware. This will allow developers to field test algorithms run against the simulator.

CHAPTER 2 MODEL DEVELOPMENT

2.1 Statement of Purpose

This project aimed to design and develop Connected Vehicle Traffic control algorithm Testing Software (CoVeTTware). The traffic simulation in this work was done by using the VISSIM simulation tool. The component object model (COM) interface was used to access the objects of the VISSIM simulation tool. The exchange of information between VISSIM and CoVeTTware was performed in real time. CoVeTTware retrieves information about vehicles at intersections from VISSIM and generates BSMs of the vehicles in real time, according to the SAE J2735 standard (Society of Automotive Engineers (SAE) (2016), Michaels et al. (2010), and Institute of Electrical and Electronics Engineers (IEEE) (2013)). The BSMs can be used by a connected vehicle traffic control algorithm to generate signal information for an intersection. The signal information includes the signal phase to be set for a signal and the period for which the signal phase should be set. The signal information is used by CoVeTTware to generate SPaT messages (SAE 2016), according to the SAE J2735 standard. Using CoVeTTware, the statuses of the signals of the intersection in VISSIM are changed in real time based on the SPaT messages,. When the period of the given phase of the signal expires, CoVeTTware resets the signal phase as UNDEFINED.

This model can be used to test any connected vehicle traffic control algorithm. Figure 2-1 shows the exchange of data among CoVeTTware, VISSIM, and the traffic control algorithm.

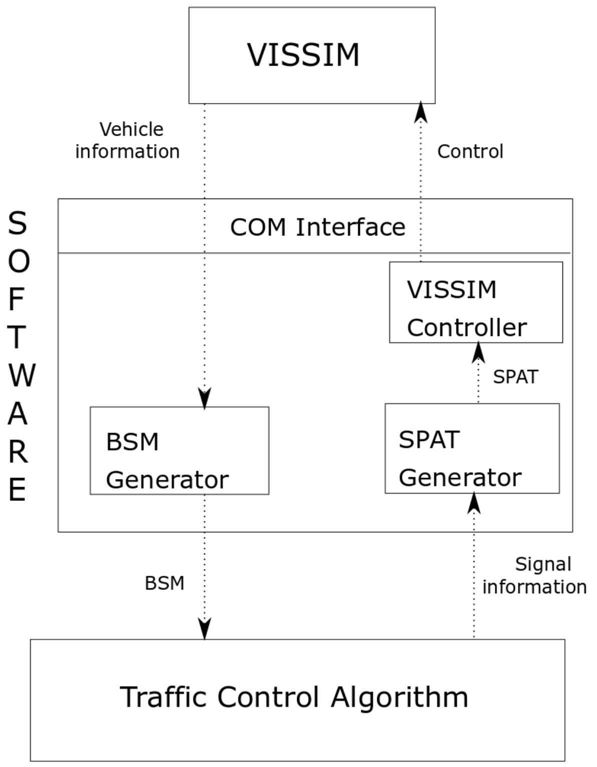

Figure 2-1 Data Exchange among CoVeTTware, VISSIM, and the Traffic Control Algorithm

2.2 Model Overview

This document explains how basic safety messages (BSMs) from vehicles can be retrieved from VISSIM using CoVeTTware. It also explains how CoVeTTware can use signal information to control the signal status of intersections in VISSIM. We assume that a signal controller exists for each intersection in the VISSIM input file. The intersection ID/number and the intersection name correspond to the signal controller number and the signal controller name, respectively. Section 2.3 explains the preliminary steps to be done. Section 2.4 documents object creation for using CoVeTTware. Section 2.5 explains the function to be invoked at the end of all operations using CoVeTTware and the functions that help in the simulation of the network.

Section 2.6 explains the BSM fields and the functions used to generate, display, and write BSM messages for the vehicles in the VISSIM network.

2.3 Preliminaries

In order to access VISSIM from Java code, the Java COM bridge (Jacob) is needed (JACOB (2021)). The following steps need to be done:

1. Register the COM server in VISSIM. See Figure 2-2.
2. Add jacob*.dll to c:\windows\system32.
3. Add jacob.jar to the library of your project.

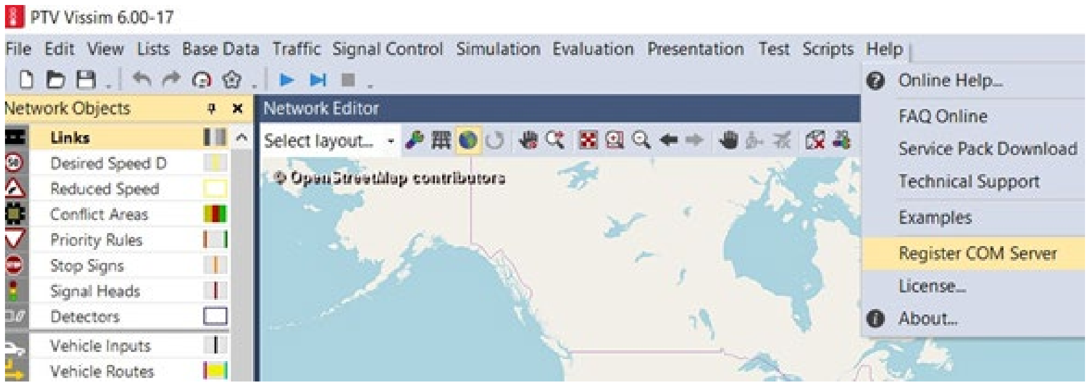

Figure 2-2 Register COM Server Command in VISSIM

To use CoVeTTware, add covettwareProject.jar to the library of your project.

2.4 Object Creation for Using CoVeTTware

To use CoVeTTware, create an object of covettware. The VISSIM input gets loaded in this process. The files for writing BSM and SPaT messages are also created here. If a file is already present, it will be overwritten.

public covettware(String VissimFilePath, String bsmFilePath, String spatFilePath)

Parameters: String VissimFilePath - The path to VISSIM input file (*.inpx).

String bsmFilePath - The path to the file where BSM messages are to be written (*.txt).

String spatFilePath - The path to the file where SPaT messages are to be written (*.txt).

The syntax for object creation of covettware is as follows:

covettware obj = new covettware(VissimFilePath, bsmFilePath, spatFilePath);

2.5 Function for Safe Closing

The function finalStep() should be used at the end of the program. It terminates VISSIM and the files (for writing the BSM and SPaT messages) safely.

public void finalStep() Parameters : No parameters Return type : void

The syntax for finalStep() is as follows:

obj.finalStep();

Other functions help in the simulation of the network.

2.5.1 Public void runVissimOnce()

This function is used to run the simulation of VISSIM by one step.

Parameters: No parameters.

Return type: void.

The syntax for this function is as follows:

obj.runVissimOnce();

2.5.2 Public int numberOfRunsFromSecs(double t)

Finds the maximum number of times the function ”runVissimOnce” should be invoked to run VISSIM for a specified time in the simulation.

Parameters: double t - Simulation time.

Return type: int - Returns the number of times the function ”runVissimOnce” should be invoked.

The syntax for this function is as follows:

int n = obj.numberOfRunsFromSecs(t);

2.6 Basic Safety Message (BSM)

The basic safety message (BSM) is a message that gives information about a vehicle in the VISSIM simulation. For each vehicle, ten BSM messages are generated per second. Section 2.6.1 explains the fields of the basic safety message. Section 2.6.2 explains the functions or methods for the BSM.

2.6.1 BSM Fields

The North, South, East, and West directions are along the positive y-axis, negative y-axis, positive x-axis, and negative x-axis, respectively. The unit is metric or imperial, depending on the selection in the VISSIM input file. The fields of the BSM are explained in table 2-1. The java class for BSM is as follows:

```
public class BSM
{
        int vehicleID;
        int messageCount;
        float currentTime;
        float latitude;
        float longitute;
        float elevation;
        float speed;
        float  heading;
        float yawRate;
        float  acceleration;
        float lateralAccelaration=0; /* unused field*/
        float verticalAcceleration=0; /* unused field*/
        int brakeOnWheels;

        int controlSystems=0; /* unused field*/ float vehicleLength;

        float vehicleWidth;

    }
```

Table 2-1 BSM Fields

Field	Type	Description	Metric Unit	Imperial Unit
vehicleID	Integer	To identify the vehicle	-	-
messageCount	Integer	The number of BSMs sent from the vehicle	-	-
currentTime	float	The time of BSM generation in the simulation	seconds	seconds
latitude	float	Distance of the vehicle North or South of x-axis (Value is positive towards North and negative towards South)	meters	feet
longitude	float	Distance of the vehicle East or West of y-axis (Value is positive towards East and negative towards West)	meters	feet
elevation	float	Height of the vehicle above the surface (= Sum of heights of the road and the vehicle)	meters	feet
speed	float	Speed of the vehicle	km / h	mph
acceleration	float	Acceleration of the vehicle	m / s^2	ft / s^2
heading	float	The angle of deviation of the vehicle from North direction (Value is positive in the anti-clockwise direction from North and negative in the clockwise direction from North)	degrees	degrees
yawRate	float	The rate of change of heading	degrees / s	degrees / s
brakeOnWheels	Integer	Set as 1 if brake is applied on wheels, and 0 otherwise	-	-
vehicleLength	float	The length of the vehicle	meters	feet
vehicleWidth	float	The width of the vehicle	meters	feet

2.6.2 *Functions for BSM*

Public List<BSM> getBSMForAllVehiclesForAllIntersections()

- Fetches the BSM messages of all vehicles in the network.

- Parameters: No parameters.
- Return type: List<BSM> - Returns the list of BSM messages of all vehicles in the network.

Public List<BSM> getBSMforIntersection(String signalControllerName, double distance)

- Fetches the BSM messages within a specified distance from the signal heads of an intersection. BSM messages of all vehicles approaching the intersection, which are within the specified distance, are fetched.
- Parameters: String signalControllerName - Name of the intersection (signal controller), double distance - distance from signal heads of the intersection.
- Return type: List<BSM> - Returns the list of BSM messages that are fetched.

Public List<BSM> getBSMforIntersection(int intersectionNumber, double distance)

- Fetches the BSM messages within a specified distance from the signal heads of an intersection. BSM messages of all vehicles approaching the intersection, which are within the specified distance, are fetched.
- Parameters: int intersectionNumber - Identification number of the intersection (signal controller), double distance - distance from signal heads of the intersection.
- Return type: List<BSM> - Returns the list of BSM messages that are fetched.

Public List<BSM> getBSMforSignalHead(String signalHeadName, double distance)

- Fetches the BSM messages within a specified distance from the signal head of an intersection. BSM messages of all vehicles approaching the signal head, which are within the specified distance, are fetched.

- Parameters: String signalHeadName - Name of the signal head, double distance - distance from the signal head.

Public List<BSM> getBSMforSignalHead(int signalHeadNumber, double distance)

- Fetches the BSM messages within a specified distance from the signal head of an intersection. BSM messages of all vehicles approaching the signal head, which are within the specified distance, are fetched.
- Parameters: int signalHeadNumber - Identification number of the signal head, double distance - distance from the signal head.
- Return type: List<BSM> - Returns the list of BSM messages that are fetched.

Public void displayBSM(BSM b)

- Displays a BSM message of a vehicle.
- Parameters: BSM b - b contains a BSM message.
- Return type: void.

Public void displayListOfBSM(List<BSM> listOfBSM)

- Displays a list of BSM messages of vehicles.
- Parameters: List< BSM > listOfBSM - listOfBSM contains a list of BSM messages.
- Return type: void.

Public void writeListOfBSMtoFile(List<BSM> listOfBSM)

- Writes a list of BSM messages to the text file specified during object creation of covettware. The files of BSM are written, separated by a comma, as follows:
- vehicleID, messageCount, currentTime, latitude, longitude, elevation, heading, yawrate, speed, Acceleration, brakeOn- Wheels(1=yes,0=no), vehicleLength, vehicleWidth.

- Parameters: List< BSM > listOfBSM - listOfBSM contains a list of BSM messages.
- Return type : void.

CHAPTER 3 MODEL IMPLEMENTATION AND TESTING

3.1 Model Implementation Examples

3.1.1 Display a BSM Message

Display the fields of the BSM b.

Program 1

```
obj.displayBSM(b);
```

Sample output:

```
vehicleID:20
messageCount:126
currentTime:9.5
latitude:2901.0
longitute:2750.0
elevation:4004.84
heading:0.0
yawRate:0.0
speed:33.489956
acceleration:0.58767843
brakeOnWheels:0
vehicleLength:13.815617
vehicleWidth:6.573727
```

3.1.2 Report Movement in the Network

Report the BSMs of all the vehicles in the network for 500 seconds in simulation. Also, write the BSMs to a file.

Program 2

```
int n = obj.numberOfRunsFromSecs(500); /*calculate the number of times
runVissimOnce() function should be invoked to run the network for
500 seconds*/
for(int i=0; i<n; i++)
{
        obj.runVissimOnce();
        List<BSM>  listOfBSM  =
obj.getBSMForAllVehiclesForAllIntersections();  /*
        listOfBSM  contains  the list of BSMs of all vehicles in
network*/
        obj.displayListOfBSM(listOfBSM); /* displays the BSMs in
        listOfBSM */ obj.writeListOfBSMtoFile(listOfBSM); /* writes the
        BSMs in listOfBSM to file */
}
```

3.1.3 Report Movement of an Intersection in the Network, When Intersection Number Is Given

Assume a metric unit is used. Report the BSMs of all the vehicles within a distance of 100 meters for intersection 1, for 500 seconds in simulation. Also, write the BSMs to a file.

Program 3

```
int n = obj.numberOfRunsFromSecs(500); /*calculate the number of times
runVissimOnce() function should be invoked to run the network for 500
seconds*/
for(int i=0; i<n; i++)
{
        obj.runVissimOnce();
        List<BSM> listOfBSM = obj.getBSMforIntersection(1,100); /*
listOfBSM contains the list of BSMs of the vehicles of intersection 1
within 100 meters*/
        obj.displayListOfBSM(listOfBSM); /* displays the BSMs in
listOfBSM */ obj.writeListOfBSMtoFile(listOfBSM); /* writes the BSMs in
listOfBSM to file */
}
```

3.1.4 Report Movement of an Intersection in the Network, When Intersection Name Is Given

Assume a metric unit is used. Report the BSMs of all the vehicles within a distance of 100 meters for intersection ”I1”, for 500 seconds in simulation. Also, write the BSMs to a file.

Program 4

```
int n = obj.numberOfRunsFromSecs(500); /*calculate the number of times
runVissimOnce() function should be invoked to run the network for
500 seconds*/
for(int i=0; i<n; i++)
{
        obj.runVissimOnce();
        List<BSM> listOfBSM = obj.getBSMforIntersection("I1",100); /*
listOfBSM contains the list of BSMs of the vehicles of intersection I1
within 100 meters*/
        obj.displayListOfBSM(listOfBSM); /* displays the BSMs in
listOfBSM */ obj.writeListOfBSMtoFile(listOfBSM); /* writes the BSMs in
listOfBSM to file */
}
```

3.1.5 Report Movement of the Signal Head of an Intersection in the Network, When the Signal Head Number Is Given

Assume a metric unit is used. Report the BSMs of all the vehicles within a distance of 100 meters from signal head 1, for 500 seconds in simulation. Also, write the BSMs to a file.

Program 5

```
int n = obj.numberOfRunsFromSecs(500); /*calculate the number of times
runVissimOnce() function should be invoked to run the network for
500 seconds*/
for(int i=0; i<n; i++)
{
        obj.runVissimOnce();
        List<BSM> listOfBSM =  obj.getBSMforSignalHead(1,100); /*
listOfBSM contains the list of BSMs of the vehicles of signal head 1
within 100 meters*/
        obj.displayListOfBSM(listOfBSM); /* displays the BSMs in
listOfBSM */ obj.writeListOfBSMtoFile(listOfBSM); /* writes the BSMs in
listOfBSM to file */
}
```

3.1.6 Report Movement of the Signal Head of an Intersection in the Network, When the Signal Head Name Is Given

Assume a metric unit is used. Report the BSMs of all the vehicles within a distance of 100 meters from the signal head ”SH1” for 500 seconds in simulation. Also, write the BSMs to a file.

Program 6

```
int n = obj.numberOfRunsFromSecs(500); /*calculate the number of times
runVissimOnce() function should be invoked to run the network for
500 seconds*/
for(int i=0; i<n; i++)
{
        obj.runVissimOnce();
        List<BSM> listOfBSM = obj.getBSMforSignalHead("SH1",100); /*
listOfBSM contains the list of BSMs of the vehicles of signal head
SH1 within 100 meters*/
        obj.displayListOfBSM(listOfBSM); /* displays the BSMs in
listOfBSM */ obj.writeListOfBSMtoFile(listOfBSM); /* writes the BSMs in
listOfBSM to file */
}
```

3.2 Signal Phasing and Timing Message (SPaT)

The signal phasing and timing message (SPaT) is a message that gives information about the phase (state) and the period for which the phase remains for the signal of an intersection. The following sections explain the fields and functions of SPaT.

3.2.1 SPaT Fields

The java class for SPaT is :

```
public class SPAT
{
        int intersectionId;
        int messageCount;
        String status;
        ArrayList<spatSignalGroupData> spatSgData;
    }
    public class spatSignalGroupData
    {
        int signalGroupId;
        String phaseState;
        float minEndTime;
    }
```

The fields of SPaT are explained in table 3-1. The field spatSgData contains a list of spatSignalGroupData. The class spatSignalGroupData contains the signal phasing and timing data of a signal group. The fields of spatSignal- GroupData are explained in table 3-2. The field ”phaseState” of spatSignalGroupData can be any valid state in VISSIM, such as RED, GREEN, AMBER, and so on.

Table 3-1 SPaT Fields

Field	Type	Description	Unit
intersectionId	Integer	To identify the intersection	-
messageCount	Integer	The number of SPaTs sent from the signal group	-
status	String	The status of the intersection	-
spatSgData	ArrayList	Contains the list of phasing and timing data for signal groups	-

Table 3-2 SPaT Signal Group Data fields

Field	Type	Description	Unit
signalGroupId	Integer	To identify the signal group of the intersection	-
phaseState	String	The phase of the signal group	-
minEndTime	float	Conveys the earliest time possible at which the phase could change	Seconds

3.3 Functions for SPaT

3.3.1 Public spatSignalGroupData createSPATSignalGroupData(int signalGroupId, String phaseState, float minEnd- Time)

This function creates the signal phasing and timing data for a signal group of an intersection.

Parameters: int signalGroupId - Identification number of the signal group, String phaseState - state of signal group, float minEndTime - simulation time in seconds.

Return type: spatSignalGroupData - Returns the signal phasing and timing data for a signal group.

3.3.2 Public SPAT createSPAT(int intersectionNumber, String intersecStatus, List<spatSignalGroupData> sgData)

The function creates a SPaT message for a signal group of an intersection (signal controller).

Parameters: int intersectionNumber - Identification number of intersection (signal controller), String intersecStatus - Status of the intersection, List<spatSignalGroupData> sgData - List of spatSignalGroupData.

Return type: SPAT - Returns the SPaT message created.

3.3.3 Public SPAT createSPAT(String signalControllerName, String intersecStatus, List<spatSignalGroupData> sg- Data)

The function creates a SPaT message for a signal group of an intersection (signal controller).

Parameters : String signalControllerName - Name of intersection (signal controller), String intersecStatus - Status of the intersection, List<spatSignalGroupData> sgData - List of spatSignalGroupData.

Return type : SPAT - Returns the SPaT message created.

3.3.4 Public void setSPAT(SPAT s)

The function sets a specified state for a specified time to the signal groups of the intersection based on the information specified in the SPaT message. After the specified time, the signal groups are reset to UNDEFINED state.

Parameters: SPAT s - s contains a SPaT message.

Return type: void.

3.3.5 Public void displaySPAT(SPAT s)

Displays a SPaT message from a signal group of an intersection.

Parameters: SPAT s - s contains a SPaT message.

Return type: void.

3.3.6 Public void writeSPAT(SPAT s)

Writes a SPaT message to the text file specified during object creation of covettware. The fields of SPaT are written separated by a comma, as follows:

intersectionNumber, messageCount, intersectionStatus, List of {signalGroupNumber, minimumEndTime, state}

Parameters: SPAT s - s contains a SPaT message.

Return type: void.

3.4 SPaT Implementation Example

3.4.1 Control Signals of an Intersection in the Network, When Intersection Number Is Given

Create a SPaT message for intersection 1 with status = FIXED_TIME_OPERATION, with signal phasing and timing for signal groups as follows.

signalGroupId: 1, State: GREEN, minEndTime: 10 seconds in simulation signalGroupId: 2, State: AMBER, minEndTime: 10 seconds in simulation signalGroupId: 3, State: REDAMBER, minEndTime: 10 seconds in simulation signalGroupId: 4, State: RED, minEndTime: 10 seconds in simulation.

Set the signal groups according to the SPaT message. Display the SPaT message. Also, write the SPaT message to a file.

Program 7

```
spatSignalGroupData d1 = obj.createSPATSignalGroupData(1,
"GREEN", 10); spatSignalGroupData d2 =
obj.createSPATSignalGroupData(2,"AMBER", 10); spatSignalGroupData d3 =
obj.createSPATSignalGroupData(3, "REDAMBER", 10); spatSignalGroupData
d4 = obj.createSPATSignalGroupData(4,"RED", 10);
```

SPAT s = obj.createSPAT(1, "FIXED_TIME_OPERATION", Arrays.asList(d1,d2,d3,d4)); /* SPAT message is created */

obj.setSPAT(s); /* set states of signal groups according to SPAT message */ obj.displaySPAT(s); /* displays SPAT message */

obj.writeSPAT(s); /* writes the SPAT message to file */

<u>Sample Output</u>

```
intersectionNumber: 1
messageCount: 1
intersection status: FIXED_TIME_OPERATION
signalGroupNumber: 1, State: GREEN, minEndTime: 10.0
signalGroupNumber: 2, State: AMBER, minEndTime: 10.0
signalGroupNumber: 3, State: REDAMBER, minEndTime: 10.0
signalGroupNumber: 4, State: RED, minEndTime: 10.0 The content written to
file is:
        1,1,FIXED_TIME_OPERATION,{1,GREEN,10.0},{2,AMBER,10.0},{3,
REDAMBER,10.0},{4,RED,10.0}
```

3.4.2 Control Signals of an Intersection in the Network, When Intersection Name Is Given

Create a SPaT message for intersection ”I1” with status = FIXED_TIME_OPERATION, with signal phasing and timing for signal groups as follows.

signalGroupId: 1, State: GREEN, minEndTime: 10 seconds in simulation signalGroupId: 2, State: AMBER, minEndTime: 10 seconds in simulation signalGroupId: 3, State: REDAMBER, minEndTime: 10 seconds in simulation signalGroupId: 4, State: RED, minEndTime: 10 seconds in simulation.

Set the signal groups according to the SPaT message. Display the SPaT message. Also, write the SPaT message to a file.

Program 8

```
          spatSignalGroupData  d1  =  obj.createSPATSignalGroupData(1,
"GREEN",  10); spatSignalGroupData  d2  =
obj.createSPATSignalGroupData(2,"AMBER",  10); spatSignalGroupData  d3  =
obj.createSPATSignalGroupData(3,  "REDAMBER",  10); spatSignalGroupData
d4  =  obj.createSPATSignalGroupData(4,"RED",  10);
           SPAT  s  =  obj.createSPAT("I1",  "FIXED_TIME_OPERATION",
Arrays.asList(d1,d2,d3,d4));  /*  SPAT  message is created */
          obj.setSPAT(s); /* set states of signal groups according to SPAT message
*/ obj.displaySPAT(s); /* displays SPAT message */
          obj.writeSPAT(s); /* writes the SPAT message to file */
```

3.5 Testing of a Connected Vehicle Traffic Control Algorithm

The steps to be followed to test any connected vehicle traffic control algorithm are as follow.

1. Add jacob.jar and covettwareProject.jar to the library of your project.
2. Create an object of covettware using the function in Section 2.4.
3. Generate BSMs using the functions described in Section 2.6.
4. Use the BSMs in the connected vehicle traffic control algorithm to generate signal status data.
5. Generate the SPaT messages using the functions described in Section 3.2.
6. Invoke the function finalStep explained in Section 2.5.

3.5.1 *An Example of a Fixed-Time Signal Generation Algorithm*

Let testInputGeneration.inpx be the VISSIM input file. Let BSM.txt and SPAT.txt be the files to which BSM and SPaT messages are to be written. Assume VISSIM input consists of five intersections with four signal groups 1, 2, 3, and 4. Create a fixed-time signal generator where the signal status changes every 20 seconds in the simulation. The signal states should be GREEN, AMBER, REDAMBER, and RED. The simulation should run for 5000 seconds in simulation. Display the SPaT messages and also write to a file.

Program 9

```
public class ConnectedVehicles
{
public static void main(String[] args)
{
        covettware obj = new covettware("testInputGeneration.inpx",
"BSM.txt","SPAT.txt"); /* creates an object of covettware */
        int n = obj.numberOfRunsFromSecs(5000); /*calculate the number of
times runVissimOnce() function should be invoked to run the network
for 5000 seconds*/
        int x = obj.numberOfRunsFromSecs(20); /*calculate the number of
times runVissimOnce() function should be invoked to run the network
for 20 seconds*/
         SPAT s;
        int sgGreen = 0, sgAmber = 1, sgRedAmber = 2, sgRed = 3;
        for(int i=0; i<n; i++)
{
```

```
obj.runVissimOnce();

if(i%x==0) /* condition is true every 20 simulation seconds */

{

for(int inters = 1; inters<=5; inters++) /* loops each intersection */

{

          spatSignalGroupData  d1  =

obj.createSPATSignalGroupData(sgGreen+1,  "GREEN",  20);

spatSignalGroupData  d2  =

obj.createSPATSignalGroupData(sgAmber+1,"AMBER",  20);

spatSignalGroupData  d3  =

obj.createSPATSignalGroupData(sgRedAmber+1,  "REDAMBER",

          20);

          spatSignalGroupData  d4  =

obj.createSPATSignalGroupData(sgRed+1,"RED",  20);

          s  =  obj.createSPAT(inters,  "FIXED_TIME_OPERATION",

Arrays.asList(d1,d2,d3,d4));

/* SPAT object created*/

          obj.setSPAT(s); /* set states of signal groups */

          obj.displaySPAT(s); /* displays SPAT */

          obj.writeSPAT(s); /* writes SPAT to file */

          }

          sgGreen  =  (sgGreen   +   1)%4; sgAmber = (sgAmber + 1)%4;

sgRedAmber = (sgRedAmber + 1)%4; sgRed = (sgRed + 1)%4;
```

```
                    }
                }
                    obj.finalStep(); /* safe closing of VISSIM and files */
            }
            }
```

CHAPTER 4 CONCLUSIONS AND RECOMMENDATIONS

The software was unit tested and also tested on some simple control problems accessed entirely through the API. A manual describing the API was written. The next steps would be a more thorough test by implementing some known control algorithms and to further verify that the messages are standards-compliant by comparing the data produced and consumed with standards via a process of code reading. Although the original design of the project included integration testing with hardware, the untimely loss of the one of the principal investigators precluded that. We recommend that at least two new "customers" for the APIs be found and encouraged to use the code and provide feedback. We also would like to see a publication of these APIs in a place that is visible to the field. With the coming age of connected vehicles, we believe these APIs will be a valuable extension to micro-simulation software. As such, we believe these results will enable connected vehicle research.

CHAPTER 5 REFERENCES

American Society for Testing and Materials (ASTM). “Standard Specification for Telecommunications and Information Exchange Between Roadside and Vehicle Systems - 5 GHz Band Dedicated Short Range Communications (DSRC) Medium Access Control (MAC) and Physical Layer (PHY) Specifications”, ASTM E2213-03, 2010.

Heckendorn, Robert B., and Eapen, Neeta A. "Connected Vehicle Traffic control algorithm Testing Software (CoVeTTware) user manual." National Institute for Advanced Transportation Technology (NIATT), University of Idaho, Moscow, Idaho, U.S.A. (2021).

IEEE. (2013), “IEEE Standard for Wireless Access in Vehicular Environments – Security Services for Applications and Management Messages”, IEEE Standard 1609:2TM, 2013.

JACOB: a JAVA-COM Bridge project, Github, 2021, https://github.com/freemansoft/jacob-project, accessed Mar 29, 2021

Michaels, C. D. R. S., D. Kelley, R. Sumner, S. Chriss, and D. Suz. "DSRC Implementation Guide A guide to users of SAE J2735 message sets over DSRC." *SAE International* (2010).

Society of Automotive Engineers (SAE) V2X Core Technical Committee. "Dedicated Short Range Communications (DSRC) Message Set Dictionary: A March 2016 Update." J2735_201603, https://saemobilus. sae. org/content/j2735_201603, accessed: October (2020).