Connected **Ve**hicle **T**raffic control algorithm **T**esting Soft**ware** (**CoVeTTware**) using VISSIM tool, BSM and SPAT messages

Neeta A Eapen, Robert B Heckendorn

---

1. Grant Information

This work is supported by PACTRANS under grant EN1828.

2. Statement of Purpose

This project aims to design and develop **Co**nnected **Ve**hicle **T**raffic control algorithm **T**esting Soft**ware** (**CoVeTTware**). The traffic simulation in this work is done using the VISSIM simulation tool. The COM interface is used to access the objects of the VISSIM simulation tool. The exchange of information between VISSIM and CoVeTTware is performed in real-time. CoVeTTware retrieves the information of vehicles at intersections from VISSIM and generates the Basic Safety Messages (BSMs) of the vehicles in real-time, according to the SAE J2735 standard. The BSMs can be used by a connected vehicle traffic control algorithm to generate the signal information for the intersections. The signal information includes the signal phase to be set for a signal and the time for which the signal phase should be set. The signal information is used by CoVeTTware to generate the Signal Phasing and Timing (SPAT) messages, according to the SAE J2735 standard. The signal status of the signals of the intersections in VISSIM is changed in real-time based on the SPAT messages, using CoVeTTware. When the time of the given phase of the signal expires, CoVeTTware re-sets the signal phase as *UNDEFINED*. This model can be used to test any connected vehicle traffic control algorithm. Figure 1 shows the exchange of data between CoVeTTware, VISSIM, and the traffic control algorithm.

3. Overview

This document explains how the Basic Safety Message (BSM) of vehicles can be retrieved from VISSIM using CoVeTTware. It also explains how signal information is used to control the signal status of intersections in VISSIM, using CoVeTTware. We assume that there exists a signal controller for each intersection in the VISSIM input file. The intersection ID/number and the intersection name corresponds to the signal controller number and the signal controller

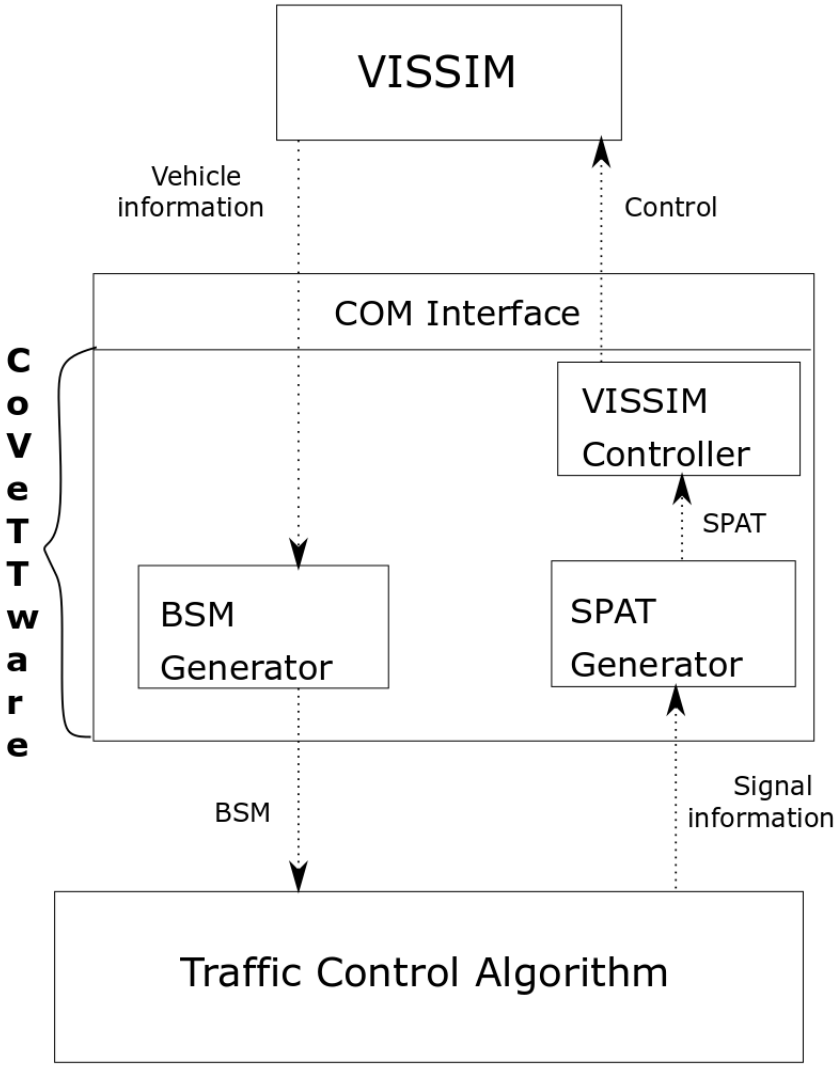

Figure 1: Data exchange between CoVeTTware, VISSIM,and the traffic control algorithm

name, respectively. Section 4 explains the preliminary steps to be done. Section 5 explains the object to be created to use CoVeTTware. Section 6 explains the function to be invoked at the end of all operations using CoVeTTware. Section 7 explains the functions which help in the simulation of the network. Section 8 explains the BSM fields and the functions used to generate, display, and write BSM messages of the vehicles in the VISSIM network. Section 9 explains the SPAT fields, and the functions used to generate, display, and write SPAT messages of intersections in the VISSIM network. Section 10 explains the steps to be followed to test any connected vehicle traffic control algorithm.

4. Preliminaries

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

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

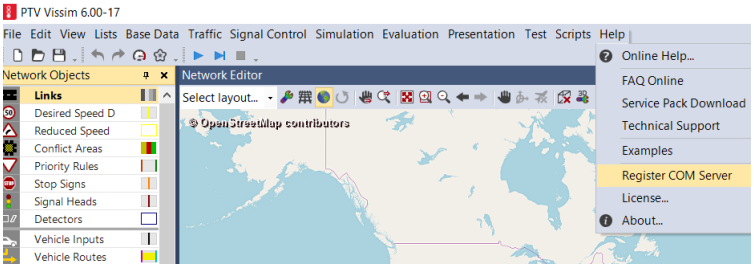

Figure 2: Register COM server

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

5. 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 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);
```

6. 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();
```

7. Functions related to simulation

This section explains the functions which help in the simulation of the network.

7.1. 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();
```

7.2. 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);
```

8. Basic Safety Message (BSM)

Basic Safety Message (BSM) is a message which gives information about a vehicle in the VISSIM simulation. For each vehicle, ten BSM messages are generated per second. Section 8.1 explains the fields of basic safety message. Section 8.2 explains the functions or methods for BSM.

8.1. *BSM fields*

The North, South, East, and West directions are along 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 BSM are explained in Table 1. The java class for BSM is :

```
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;
}
```

8.2. *Functions for BSM*

8.2.1. *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.

8.2.2. *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 which are fetched

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 vehicles	meters	feet

Table 1: BSM fields

8.2.3. 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 which are fetched

8.2.4. public List<BSM> getBSMforSignalHead(String signalHeadName, double distance)

Fetches the BSM messages within a specified distance from a 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
Return type: List<BSM> - Returns the list of BSM messages which are fetched

8.2.5. public List<BSM> getBSMforSignalHead(int signalHeadNumber, double distance)

Fetches the BSM messages within a specified distance from a 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 which are fetched

8.2.6. public void displayBSM(BSM b)

Displays a BSM message of a vehicle.

Parameters: BSM b - b contains a BSM message

Return type: void

8.2.7. 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

8.2.8. 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 , as follows:

vehicleID, messageCount, currentTime, latitude, longitude, elevation, heading, yawrate, speed, Acceleration, brakeOnWheels(1=yes,0=no), vehicleLength, vehicleWidth

Parameters: List< BSM > listOfBSM - listOfBSM contains a list of BSM messages

Return type : void

8.3. Examples

8.3.1. 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

8.3.2. 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 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 */
}
```

8.3.3. Report movement of an intersection in the network, when intersection number is given

Assume 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 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 */
}
```

8.3.4. Report movement of an intersection in the network, when intersection name is given

Assume 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 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 */
}
```

8.3.5. Report movement of the signal head of an intersection in the network, when the signal head number is given

Assume 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 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 */
}
```

8.3.6. Report movement of the signal head of an intersection in the network, when the signal head name is given

Assume 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 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 */
}
```

9. Signal Phasing And Timing Message (SPAT)

Signal Phasing And Timing Message (SPAT) is a message which gives the information about the phase (state) and the time for which the phase remains, in a signal of an intersection. Section 9.1 explains the fields of SPAT. Section 9.2 explains the functions or methods for SPAT.

9.1. 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 2. 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 spatSignalGroupData are explained in Table 3. The field "phaseState" of spatSignalGroupData can be any valid state in VISSIM like *RED, GREEN, AMBER,* and so on.

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 <spatSignalGroupData>	Contains the list of phasing and timing data for signal groups	-

Table 2: SPAT 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

Table 3: spatSignalGroupData fields

9.2. Functions for SPAT

9.2.1. public spatSignalGroupData createSPATSignalGroupData(int signalGroupId, String phaseState, float minEndTime)

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.

9.2.2. 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.

9.2.3. public SPAT createSPAT(String signalControllerName, String intersecStatus, List<spatSignalGroupData> sgData)

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.

9.2.4. 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

9.2.5. 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

9.2.6. 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 , as follows:

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

Parameters: SPAT s - s contains a SPAT message

Return type: void

9.3. Example

9.3.1. Control signals of an intersection in the network, when intersection number is given

Create 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 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 */
```

Sample Output:

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}

9.3.2. Control signals of an intersection in the network, when intersection name is given

Create 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 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 */
```

10. Testing of connected vehicle traffic control algorithm

The steps to be followed to test any connected vehicle traffic control algorithm is as follows.

- Add *jacob.jar* and *covettwareProject.jar* to the library of your project
- Create an object of *covettware* using the function in Section 5
- Generate BSMs using functions of Section 8
- Use the BSMs in the connected vehicle traffic control algorithm to generate signal status data
- Generate the SPAT messages using functions in Section 9.
- Invoke the function *finalStep* expalined in Section 6

10.1. An example of 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 5 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 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 */
    }
}
```