Assignment #4  
Distributed Components

Advanced Discrete Simulation - EPA1351

Group 14 - Daan van Bilsen (4198433)

Roel Faber (4307631)

Ioannis Papadogeorgos (4737296)

Maarten van de Kamp (4291611)

# 1. Introduction

After achieving to obtain a clean and consistent dataset in assignment 1, which was used for both simulation (in assignment 2) and visualization (in assignment 3) purposes all three assignments will culminate in a functional large-scale/distributed simulation. The simulation model built in Simio and the visualization scripts in Python will be connected to create a live visualization of the results from the simulation model. This connection between these components is achieved by reading and writing to an SQL database, which can be seen as the wrapper combining the two parts together.

This is an execution of distributed simulation, which plays an important role in complex simulation modeling and a real-time interaction with the executing model is achieved. This includes the notion of the Dynamic Data Driven Applications Systems (DDDAS) as additional data are incorporated into the executing application and at the same time, applications are enabled to dynamically steer the measurement process[[1]](#footnote-0). An effort has been made to reuse an already developed simulation model which also needs the proper interconnections with the rest of the vital components of the system. Various syntactical consistency checking are also needed in order to guarantee both a smooth transfer of data and the proper synchronization between the different components.

A communication bus has been created within this assignment, since we desired live interaction between the databases, the simulation software, and the visualization scripts. The outcomes of the simulation can be visualized as the simulation software stores data which can then immediately be accessed by our visualization scripts. Furthermore we created input scripts which write data that needs to be read by our simulation software. For this purpose MySQL is used as the database server. This setup allows for solid live interaction between python and simio data manipulation processes, including writing and reading from both sides. In this report we report more in depth on the communication setup and difficulties, explain the changes made to the visualization and simulation software compared to assignments 2 and 3 and explain how the two programs communicate with SQL and through SQL with each other.

# 2. Theory of Distributed Simulation

The simulation model of a large-scale complex system is considered as a collection of coupled simulation models. Each of these models serves a different purpose, which ultimately leads to a distribution simulation study. The workload regarding the development and the maintenance of components can be allocated to different teams, given the need for integration is not absolute. Specialised simulation tools can be coupled using ‘message buses’, which supports the realisation of a co-simulation environment and allows controller hardware-in-the-loop validation tests[[2]](#footnote-1).

However, the different components are not applicable to every situation, given that standards do not always exist or applied in order to achieve an efficient communication between them. Within the framework of organisations, packages hide the access to their internal data which are needed to interconnect the components, while their system are dependent on each other. Some components are available, but even the simulation environments vary greatly, which deteriorates their alignment. Semantically, the components can be aligned, while pragmatically the differences are spotted in terms of focus. Both aspects lead to the conclusion that the inner mechanism of the components has to be known, followed by the comprehension of the purpose they serve.

The majority of the available simulation packages do not offer a direct access to the attributes, but offer interfaces through which some of the model attributes are accessible[[3]](#footnote-2). For components which work quite independently and simulated separately, these interfaces can resemble better the reality. For this reason, distributed simulation is fruitful when different parts of an organisation are interconnected through a mechanism offered by distributed simulation architectures. This coordination unit also enables functions such as the run control of the overall system, the data sharing and the time synchronization, setting the rules to govern the overall simulation. Two different complex simulation models which are developed in different simulation packages can be thus connected or interfaced, despite the limitations posed by their basic design.

# 3. Setup

The end-user is recommended to run the simulation in the following order:

1. Ensure all the necessary documents and MySQL tables are present
2. Reset the Simio simulation
3. Restart and re-run the python kernel
4. Run the Simio simulation (do not fast forward)

First we will give information about the general structure of the two models. Afterwards we will elaborate how they communicate with each other, using theory on distributed simulations given in the lectures. In short, we build upon the simulation model of assignment 2 by adding different types of traffic, multiple sources and sinks, states and processes to read relevant data from and write data to a MySQL-server, that is connected to a python script as well.

## 3.1 Simulation Model (Simio)

We started from the simio model we instantiated using the cleaned data for the N1-road between Dhaka and Chittagong of assignment 2, which included for example a chance of bridges to breakdown and a processing time dependent on bridgelength. To be able to communicate with MySQL, first a connection was established with the DbConnect1-element, according to the information mentioned in appendix A. This appendix also gives an overview of all tables used in MySQL that are referred to.

At the beginning of a run a process starts: ‘OnRunInitialized’, that:

* Deletes existing data from the database,
* Searches bad bridges in Simio (BadBridgeX: properties as controls to assign broken bridges in Simio) and
* Reads bad bridges from MySQL (to SQLBadBridge\_X: the states of the model in Simio storing broken bridges imported from MySQL)
* After that, each bridge goes through a process of breaking bridges down, dependent on the specified probability and on the bridges read from MySQL. These processes are called ‘TypeX\_RunInitialized’, where X is A, B, C or D, dependent on bridge quality.

To be able to visualize different kinds of traffic we incorporated several different types of model entities. Each entity represents a part of the overall traffic. Based on the AADT data, three entity types were made:

* Cars\_etc (including Utility, Car, Auto, Rickshaw, Motor Cycle, Bi-Cycle, and Cycle Rickshaw),
* Trucks (heavy, medium and small) and
* Busses (large, medium and micro). They all have the same speed: 48 km/h.

Another substantial change we made to this model was adding sources and sinks along the route to align the traffic data in our model with traffic densities as given in the AADT data for all different types of traffic. In order to do so we calculated interarrivaltimes at the sources, and calculated the percentage of entities that should continue on the road when approaching a sink (see the Traffic2-excel file for more details). The interarrivaltimes are loaded into the model in Simio with the ‘SimioSpreadsheetAss4\_part2’-file. This file also makes sure connections with added sources and sinks are established and that the basic nodes connecting the N1 to the sinks got a LinkRule that assigns modelentities to base their destination on the linkweights. These weights are dependent on the calculated percentage of entities that should continue at a sink.

Next to that, several segments were created, to be able to store and compare traveltimes of modelentities. Therefore, 22 states for a modelentity were created: SegmentX\_LRP###, to store the times at specific points in the model. Also, 22 tallies that store the time it takes for modelentities to travel from the beginning to the end of a segment were created (Time\_in\_segmentX), which is also included in the ‘SimioSpreadsheetAss4\_part2’-file loaded into simio.

Also, a subclass of a path was created (MyPath), with three states: Cars per hour, Trucks per hour and Busses per hour, ‘counting’ the amount of vehicle per hour on the road. Each entity when at the end of a road goes through a process (Road\_exit) that adds ‘himself’ to the amount of that specific type on that road.

Each hour, a process is started (Timer1) that:

* Writes the amount of vehicles that crossed each segment to the ‘roads’-table in MySQL;
* Writes the traveltimes for each segment to the ‘traveltimes’-table;
* Repairs and/or breaks random bridges dependent on the input from MySQL;
* Start the process that writes into the ‘Semaphore’-table, which ‘hints’ to python all of the relevant data of the past hour has been written into MySQL

## 3.2 Visualization (Python) pseudo-code

The current quality of the N1 road is visualized based on the script created within the previous assignments, which will be constantly updated with the new data that Simio writes to the database. The adjusted dataframe of Python is informed when new data arrive which will be visualized. This takes place automatically every minute, where Python asks for data from Simio without using a particular exchange of an occurrence to call for data import. New data which haven’t been generated yet will be supplemented with old data. The script is read through, and the new data are stored and edited using Folium. Below one can see the detailed pseudo-code for this script.

1. Import necessary libraries
2. Import necessary data about roads, bridges and the LRP start and ends of the chosen segments
3. Connect to the mySQL connector with the username, password and database discussed in class. From this, import the data about the traffic on the roads.
4. Create a list of all the bridges. When the user is later asked about what bridge to break, this bridge needs to be in this list.
5. This cell contains two make-map functions. These functions are called up as soon as a Simio updates the data within mySQL. Both functions can be called up to create the map. The function ‘makemap1’ is the recommended function to use as it shows the delay times of the road segments. The ‘makemap2’ function on the other hand, plots the traffic density along the road. Although the user is still allowed to visualise the traffic density, it mainly serves for validation purposes. Both functions are very similar in their nature.
   1. Import the most recent traffic data
   2. Create a list called ‘normaltimes’ that indicates the times it normally takes to cross a road segment. This is subsequently used to calculate the delay times.
   3. Divide the data to the road segments in three steps
      1. For the first segment, take Row 0 until the first LRP
      2. For the intermediate segments, take the values between the LRP’s
      3. For the last segment, take the value of the last LRP until the last row
   4. Correctly place all data in a dataframe by rows
   5. Calculate the delay time (deltat) for each segment and find the appropriate colour in accordance with the delay
   6. Plot the data in a ‘folium’ map in a loop for each segment. This allows each segment to be of different colour, depending on the delay time (or traffic density if the ‘makemap2’ function is called up
   7. For each segment, allow a popup that will visualise the three traffic categories into barcharts
   8. For all the bridges that are to be broken, create a small marker with basic information about the bridge that is broken.
   9. Create an appropriate legend
   10. If the PNG\_Files folder does not exist yet, create the folder. All output will be saved in ‘screenshot’ format here, from which eventually a ‘gif’ can be created
   11. If the HTML\_Files folder does not exist yet, create the folder. All output will also be saved in html format, where each individual frame (hour by hour) can be revisited and more thoroughly analyzed
   12. Save the map in both folders

The cell also contains an additional function. After the simulation run, the python script will create a ‘gif’ out of the ‘screenshots’ inside the PNG\_Files folder. This allows the user to revisit all frames in ‘video’ format. It also allows the user to pause, fastforward or scroll within the ‘gif’ with the appropriate software (software package ‘gifviewer’ has been succesfully tested).

1. This cell contains functions mainly related to the mySQL connections:
   1. ‘Truncate’ removes all the data within the table and resets the Auto-Increment.
   2. The ‘writezero’ functions will write a zero in the semaphore tables
   3. ‘Writeone’ will write a one in the semaphore table for breaking a random bridge
   4. ‘Readsemaphore’ will read when Simio has placed new data in MySQL
   5. ‘Writebridge’ will write the bridges to mySQL that are to be permanently destroyed during the model run
   6. ‘Writerandombridge’ will write the bridge to mySQL that is to be broken for one hour
   7. ‘Bridgetobreak’ will prompt the user to enter the LRP of the bridge that is to be broken permanently
   8. ‘Main’ contains the main script of initialization and contains the script prompting the user how many bridges he/she likes to see broken.
2. This cell contains another prompt, asking the user whether or not he likes to see an additional bridge broken every hour
3. The user is asked what type of visualisation is preferred
4. The script becomes subsequently ‘stuck’ within a while-loop. This loop will recur until the end of the simulation. Each second, the script will see whether the ‘semaphore’ table has been changed by Simio. If so, it will start the visualisation process described in (5). It will then remove the ‘semaphore’ and write which random bridge is to be broken during this hour.
   1. If this is the first of the simulation frame, the driver is opened. With this, the output of the file is automatically opened in Google Chrome and a screenshot is taken and stored within the PNG\_Files folder
   2. Otherwise, the Google Chrome window is refreshed and the most recent output should appear.

This loop will stop after the simulation is finished, or if there have been no more changes over the past 10 minutes. The script will then create a gif of the obtained output screenshots.

## 3.3 Communication through MySQL

As mentioned before we used MySQL to allow our simulation and visualization components to communicate with each other. We created a local instance of mySQL with a user that can access a database and allowed Simio and Python to log in as this user and either read from or write to the MySQL schema. How MySQL is set up exactly is documented extensively in Appendix A. Below we will give a high-level overview of how SQL is used to communicate when our program is run:

1. First, Python is started. The python script truncates all relevant tables, deleting all rows and resetting auto-incrementing values. Python also writes user-specified bridges to SQL. Then python sets the value for the semaphore to 0 and waits for Simio to update this value to 1.
2. Simio reads the user-specified broken bridges
3. When Simio has run for one hour it writes all relevant data to SQL. When this is done, it updates the value of the semaphore to 1.
4. When Python detects a 1 in the semaphore it reads the SQL information to a pandas dataframe after which the visualization is made. When it is done it updates the semaphore to 0.
5. If the user chooses to break a random bridge during the simulation run, another semaphore table is activated. Furthermore, the Python script will place a new random bridge in the ‘writebridgerandom’ table, indicating that this is the next bridge that is to be broken.
6. When the semaphore table for random bridge destruction is activated, Simio will read which bridge to break at the start of the hour. The same bridge will also be restored by the end of the hour.

Steps 3 and 4 are repeated until the simulation is finished. When this is the case the Python file will automatically stop running after some time.

In order to assure the quality of the data, the consistency among the different components should be guaranteed and therefore syntactic and semantic errors were combated. An example of this is related to objects: when reading from MySQL the string of the name of a bridge to be broken, Simio could not store that as an ‘Object reference’. Therefore, it was stored in a string state. Later on, this string was compared to bridges-names, causing them to break down.

More detail on all values that are written to and read by Python and Simio is given below, together with the algorithms and logic implemented:

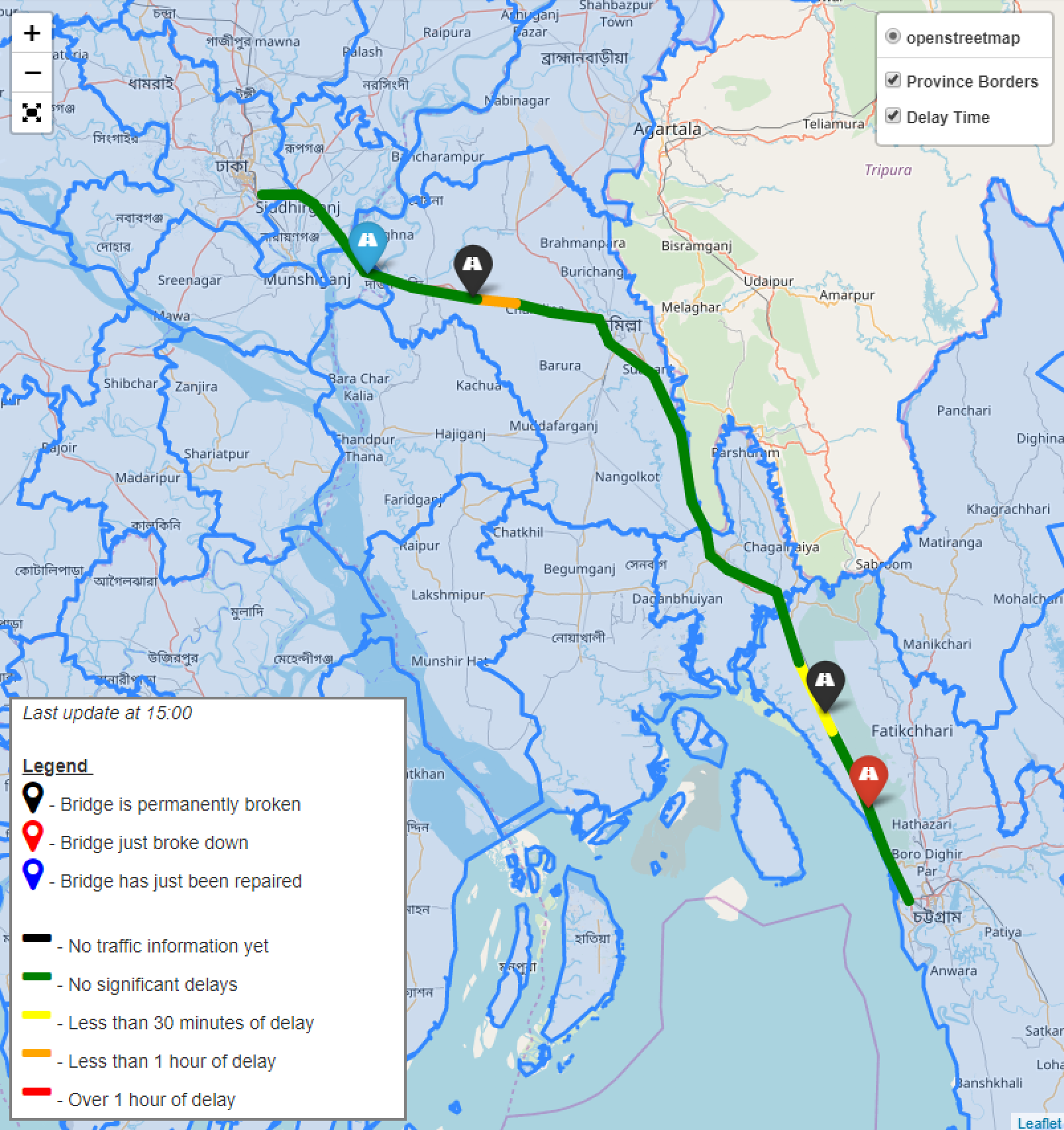
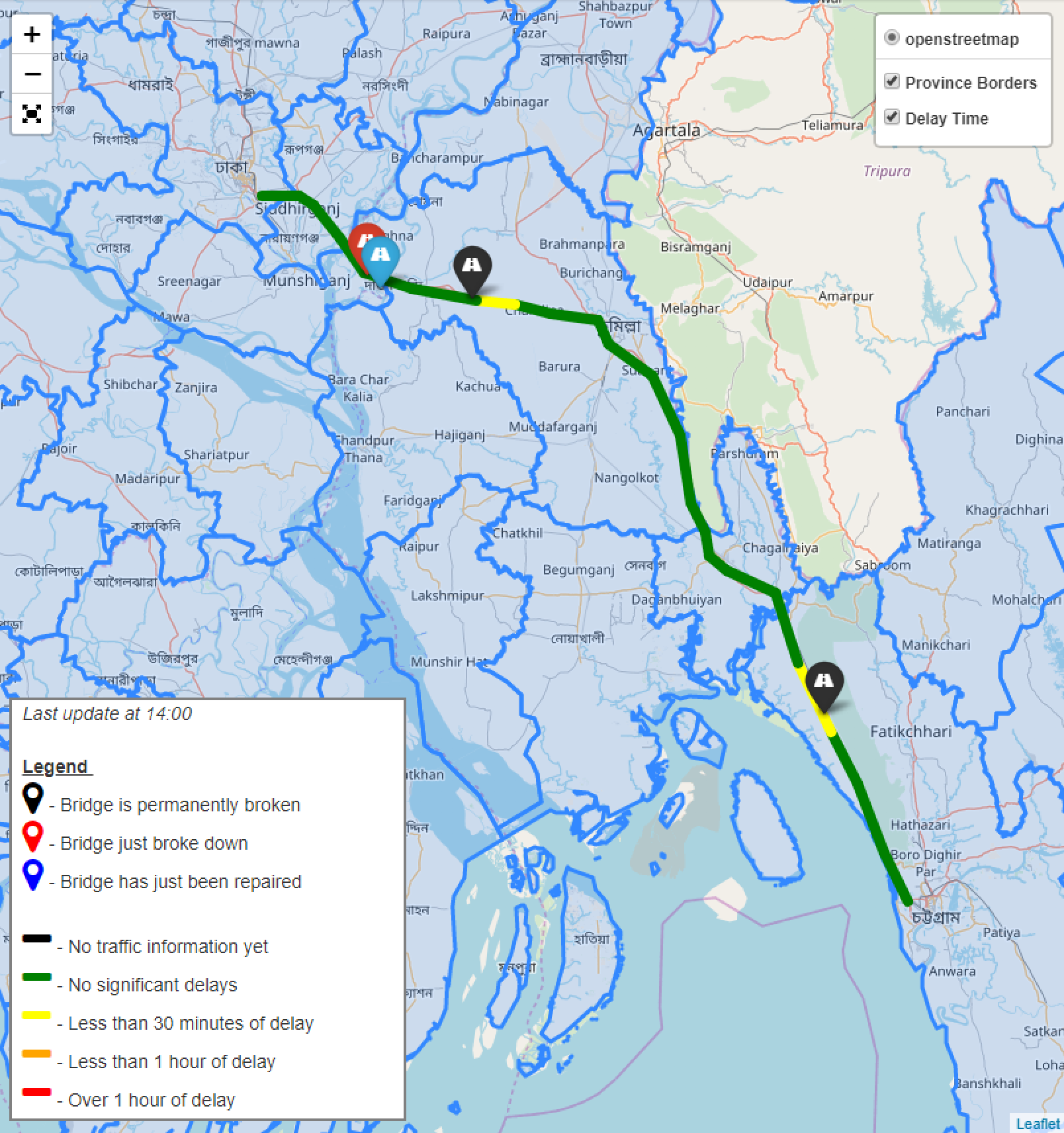
# 4. Results

The results are presented of the individual frames from our own tested simulation are presented in the folder (G14\_Own\_Output) attached. Some results are presented below.

The user is given the option to break a specific bridge permanently and/or to break random bridges on the road segment for a period of 1 hour. In the experiment below, bridges relatively far apart are broken (bridges LRP056a and LRP191b). On top of that, the random bridges are broken and fixed on an hourly basis.

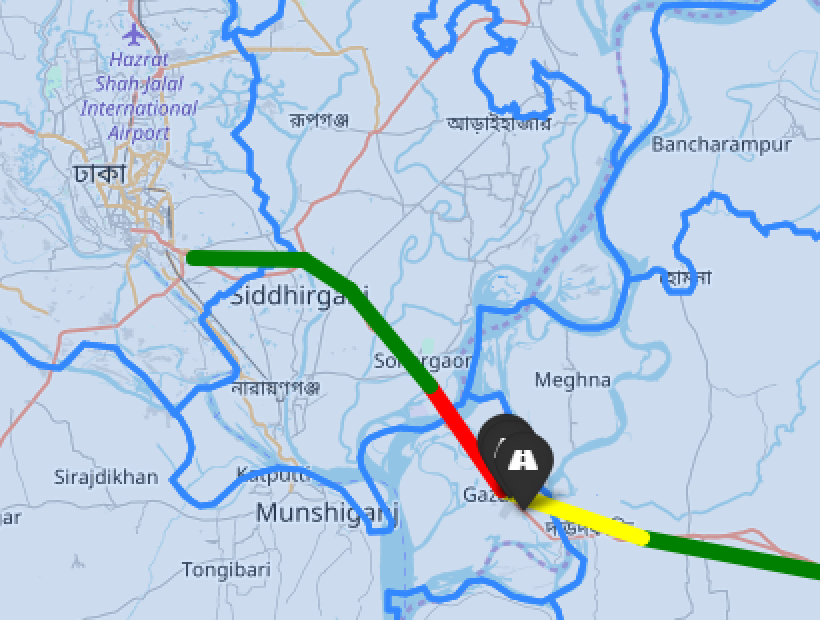
Figure 1a and 1b presented below show the results on Day 1 of the simulation at times 14:00 and 15:00. These results show that the permanently bridges have a far larger impact on delays on the road than temporarily broken bridges do. Even if the bridges are located near each other, they do not create a significant delay around their own location. Although the bridges are constantly broken all throughout the simulation, the G14\_Output\_GIF.gif’ file, also located in folder ‘G14\_Own\_Output’ further indicates that there are only two segments with any delay, both of which are at the permanently damaged bridges.

This indicates that the duration that a bridge is damaged crates a more extreme delay than the amount of bridges that are damaged. Even if two bridges near each other are damaged one after each other, there seems to be no significant delay. However, more experiments need to be run in order to confirm this hypothesis.



*Figure 1a, b: An overview of the delays at 14:00 and 15:00 respectively*

As expected however, permanently broken bridges that are geographically close to each do create extreme delays from the start. This can be seen in Figure 2 below, in which the bridges LRP031a, LRP032a and LRP033a are permanently broken.



*Figure 2: An overview of delay due to the broken bridges at LRP031a, LRP032a, LRP033a*

# 5. Conclusion

After bringing together the distributed components developed within the previous assignments, a complex large-scale simulation was created using the data which were stored by the simulation software and accessed by our visualization scripts. An SQL database as a wrapper for the interconnection of the python scripts and Simio data manipulation processes, while the additions of traffic types, states and properties as well as model objects enabled the interfacing with the MySQL server. The constant reloading of the python script guaranteed the interactive presentation of the user commands regarding bridges of his selection, whose storing in SQL facilitates the immediate update of Simio inputs. The random bridges which are broken and fixed on an hourly basis reveal the degree of impact on the transport delays over particular road segments, which is vastly affected by the duration that a bridge remains damaged. However, more experiments would need to be conducted to validate how impactful is the sequence of broken bridges in the final delay. The immediate response of the visualization script as well as its interconnectivity with the rest of the modeling components guaranteed the interactiveness of the visualization and the successful coupling of the distributed components.

**Recommendations for future research:**

* Instead of breaking a random bridge during the script, we would like to investigate the option which allows the user to interactively input the bridge to break during the simulation run.
* Future research can also focus on breaking multiple bridges during the experiment.
* The current model is still extremely static and seems only dependent on the bridges that are permanently broken.
* Running actual designed experiments to check which bridges are most vulnerable/critical.
* Adding more output parameters and options for visualizations.

**Critical Reflection**

This assignment has focussed on composability and distributed simulations. Three software packages were used in order to set up, calculate and visualise a simulation about the traffic within Bangladesh. We believe we were very successful in creating a suitable model for the assignment at hand. The imported code is first implemented into python, from which data is ascribed to Simio through MySQL, the wrapper which is responsible for the data sharing between the two packages. The user can choose up to three bridges which are to be permanently damaged, but this could in theory be extended to an unlimited amount. Furthermore, there is constant interaction between the Simio input and the Python output. The visualisation script is able to run and thereby visualise in mere seconds, which is well within the time of the Simio simulation. On top of that, the Simio model is able to read and write the newly ascribed data/updated data (for example, the next random bridge to break) and perform this task in even less time.

Still there are significant improvements to be made to the model. The largest issue concerns the time it takes for the entire simulation to run (nearly 2 hours). Both packages run a lot faster if they are run individually and ideally, the Simio package should have a function/process that can pause and run the simulation as a whole. Hereby, Simio can constantly fast-forward its simulation and pause on the hour mark, which in turn allows the python script to run much faster. The current work only allows python to be paused. We firmly believe that the features included ‘Recommendations for future research’ could be implemented within a small amount of time in order to get the full effect of the capabilities of distributed modeling.

# Appendix A: Setting up SQL

For the communication between our simulation and our visualization software MySQL databases were used. It is absolutely vital that the settings in MySQL are correct: otherwise the programs can not read from these tables. For this an SQL back-up (dump) is delivered together with this report. Should this dump not work or not be available to you we have written a description of our schema so that it can be recreated manually. In an ideal world we’d have written a script that automatically checks the SQL set-up and changes it in Python.

We set up a local instance of MySQL following the outlines given in the assignment 4 documentation. We have a local user ‘epa1351user’ with password xgt65RR##. This user has access to all relevant databases. We set up one main schema: epa1351group14. Below we give the names of all tables in this schema, together with the columns, datatypes and other necessary information (primary key, auto increment etc.)

|  |  |  |
| --- | --- | --- |
| **Table = ‘roads’** | | |
| **Columns** | **Datatype** | **PK/NN/AI etc.** |
| ID | INT(11) | Primary Key, Not Null |
| RoadName | VARCHAR(45) | - |
| AmountCars | DOUBLE | - |
| AmountTrucks | DOUBLE | - |
| AmountBusses | DOUBLE | - |

|  |  |  |
| --- | --- | --- |
| **Table = ‘brokenbridge’** | | |
| **Columns** | **Datatype** | **PK/NN/AI etc.** |
| bridgeid | VARCHAR(45) | Primary Key, Not Null |
| idbroken\_bridge | VARCHAR(45) | - |

|  |  |  |
| --- | --- | --- |
| **Table = ‘timesegment’** | | |
| **Columns** | **Datatype** | **PK/NN/AI etc.** |
| TimeSegment | FLOAT | - |
| SegmentID | INT(11) | Primary Key, Not Null |

**This is a pretty long table. For the sake of brevity we took out a couple columns. The columns are simply labelled Segment1; Segment2; Segment 3; etc. until Segment22 which is the last segment. They are all of the same datatype.**

|  |  |  |
| --- | --- | --- |
| **Table = ‘traveltimes’** | | |
| **Columns** | **Datatype** | **PK/NN/AI etc.** |
| Timestep | INT(11) | Primary Key, Not Null |
| Segment1 | VARCHAR(45) | - |
| Segment2 | VARCHAR(45) | - |
| Segment3 | VARCHAR(45) | - |
| ... | VARCHAR(45) | - |
| Segment20 | VARCHAR(45) | - |
| Segment21 | VARCHAR(45) | - |
| Segment22 | VARCHAR(45) | - |

|  |  |  |
| --- | --- | --- |
| **Table = ‘writebridge’** | | |
| **Columns** | **Datatype** | **PK/NN/AI etc.** |
| bridgeid | VARCHAR(45) | - |
| id | INT(11) | Primary Key, Not Null, Auto Increment |
| bool | INT(11) |  |

|  |  |  |
| --- | --- | --- |
| **Table = ‘writebridgerandom’** | | |
| **Columns** | **Datatype** | **PK/NN/AI etc.** |
| bridgeid | VARCHAR(45) | - |
| id | INT(11) | Primary Key, Not Null, Auto Increment |
| bool | INT(11) |  |

|  |  |  |
| --- | --- | --- |
| **Table = ‘semaphore’** | | |
| **Columns** | **Datatype** | **PK/NN/AI etc.** |
| Semaphore | INT(11) | Primary Key, Not Null |

|  |  |  |
| --- | --- | --- |
| **Table = ‘semaphore\_randombridge’** | | |
| **Columns** | **Datatype** | **PK/NN/AI etc.** |
| SemaphoreRandomBridge | INT(11) | Primary Key, Not Null |
| SemID | VARCHAR(45) | - |

We hope that everything is clear and our live visualization can be replicated successfully!

1. Darema, F. (2005). Grid computing and beyond: The context of dynamic data driven applications systems. *Proceedings of the IEEE*, *93*(3), 692-697. [↑](#footnote-ref-0)
2. Mosshammer, R., Kupzog, F., Faschang, M., & Stifter, M. (2013, November). Loose coupling architecture for co-simulation of heterogeneous components. In Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE (pp. 7570-7575) [↑](#footnote-ref-1)
3. Boer, C. A., & Verbraeck, A. (2003, December). Distributed simulation and manufacturing: distributed simulation with cots simulation packages. In Proceedings of the 35th conference on Winter simulation: driving innovation (pp. 829-837). Winter Simulation Conference. [↑](#footnote-ref-2)