

Network design of a bicycle-sharing system

Silvana de Leeuw (370701)

July 6, 2015

Abstract

In this report we will be analysing a bicycle-sharing system. If we want to stimulate people to use these systems, we first need to make sure that people have trust in the system. We will be validating two different models that estimate the performance of a bicycle-sharing system with a simulation. We will also use the simulation program Anylogic to gain insight into the system. After the validation we will implement a case study and conclude that the performance of the Equilibrium State Model seems to improve when the system contains more stations and bicycles while the Stochastic Network Flow model is not capable of estimating all cases.

1 Introduction

Bicycle-sharing systems are commonly used in big cities. In our view there are two main reasons why these systems should be used more. Firstly transportation by bicycle is a green way of transportation as it does not emission CO_2 and if the bicycles are shared, less have to be made within a factory. Secondly it is also healthier for humans to use a bicycle instead of a car or public transport. If we want more people to use the system, we need to make sure that the bicycles are at the right station at the right time such that the service level of the system is high enough to pay for the bicycles, the redistribution, and the docking stations.

If the number of bicycles, the redistribution, and the renting price of a bicycles are chosen optimal, this system could generate profits. If we want to know which redistribution should take place, we first need to know more about the bicycle movements in the system. Besides, if we want to make it profitable, we need to be able to set a requirement utilization for each bicycle.

In a bicycle-sharing system, bicycles are shared and each bicycle can be docked at one of the stations in the system. People who take a bicycle from a docking station need to dock it at another station and pay for their trip. Now someone else can use the same bicycle from the station it was docked. In a bicycle-sharing system the number of trips made needs to be maximized since these trips generate revenues. A trip can only be made if there is a bicycle available at a station when a customer arrives. We would like to verify some results from Shu, Chou, Liu, Teo and Wang [1]. In this article several models related to bicycle-sharing systems, are constructed and verified with a simulation. These models aim to estimate the performance of bicycle-sharing system. They estimate the number of trips made over the total time period and the bicycle utilization with a optimal initial bicycle distribution in a Stochastic Network Flow Model. They also provide an Equilibrium State Model to determine where the bicycles will be when time goes to infinity. With this model, we can gather information that can be useful for determining the redistribution of the bicycles after some time periods.

In this report we will firstly verifying the Stochastic Network Flow Model and the Equilibrium State Model constructed in Shu et al [1] with a numerical example and a case study. We will use two simulations to compare the models with. Secondly we will compare the performance of the models with the small numerical example with the performance of the model with the data from the case study. Are the Stochastic Network Flow Model and the Equilibrium State Model more accurate when the bicycle-sharing system contains more docking stations and bicycles?

In section 2 the Stochastic Network Flow and Equilibrium State Model are described with the necessary lemma's and proofs. Then in section 3 the numerical example is introduced and we explain how we verify the models. In section 4 we will see the performance of both models with a bicycle-sharing systems that contains more stations and bicycles. In section 5 we will conclude the report and finally in section 6 some suggestions for future research and discuss the findings.

2 Model Description

In this section we will describe two models for bicycle-sharing systems constructed in Shu et al [1]. To analyse these models, we will first look at the connections between the customer arrival rates and the expected number of trips that will take place.

2.1 Relationship between customer arrivals and trips made

In order to construct a model to estimate the performance of a bicycle-sharing system, Shu et al [1] firstly describe connections between the expected number of trips made and the customer arrival rate for a certain origin-destination trip. They used three lemma's to construct their models. To describe these lemma's we will firstly define some sets, parameters and variables.

- **Set** ζ : the set of all stations in the bicycle-sharing system.
- **Set** T : the set of all K time periods (1...K).
- **Parameter** $r_{ij}(t)$: the customer arrival rate at station i of customers with destination station j during time period t .
- **Parameter** $D_{ij}(t)$: the number of customer arrivals traveling from station i to station j during time period t .
- **Parameter** $D_i(t)$: the number of customer arrivals at station i during time period t .
- **Variable** $y_{ij}(t)$: the expected number of bicycles moving from station i to station j during time period t . Note that y_{ii} denotes the number of bicycles that stay at station i during time period t .
- **Variable** $y_i(t)$: the expected number of available bicycles at station i at the beginning of time period t .
- **Parameter** $x_i(t)$: the number of bicycles available at station i at the beginning of time period t .

Using these definitions we can define the number of bicycles that leaves station i during time period t as $\min(x_i(t), D_i(t))$. If $x_i(t)$ is smaller than $D_i(t)$ we assume that the first arriving customers will use the available bicycles. If a customer arrives, but there is no bicycle available, then the customer will be declined and leave the system.

Let $Q_i(p)$ be a sequence of independent Bernoulli random variables with mean p while $0 \leq p \leq 1$, then $D_i(t)[p] = \sum_{k=1}^{D_i(t)} Q_k(p)$ This means that $D_i(t)[p]$ could be seen as the number of arriving customers that is tagged when there is a chance of p to be tagged.

With the Poisson Thinning Lemma as described in Ross [2], we can conclude that $D_i(t)[p]$ is Poisson distributed with rate $p \times \sum_{j \in \zeta: j \neq i} r_{ij}(t)$. If we define $p_{ij}(t) \equiv r_{ij}(t) / \sum_{k \in \zeta: k \neq i} r_{ik}(t)$ we can state

$$D_{ij}(t) \sim D_i(t)[p_{ij}(t)]. \quad (1)$$

For a given $x_i(t)$, let

$$\min(x_i(t), D_i(t)[p]) = \sum_{k=1}^{\min(x_i(t), D_i(t))} Q_k(p). \quad (2)$$

According to the equation (2) combined with the Poisson Thinning Lemma, the number of bicycles leaving station i with destination station j follows the distribution of $\min(D_i(t), x_i(t))[p_{ij}(t)]$.

Now we can define the number of bicycles available at station i at the beginning of the next time periods. This is equal to the number of bicycles available at station i at the beginning of this time period t minus the number of bicycles that depart from station i during time period t , plus the number of bicycles that arrive station i during time period t .

$$x_i(t+1) = x_i(t) - \sum_{j \in \zeta: j \neq i} \min(D_i(t), x_i(t))[p_{ij}(t)] + \sum_{j \in \zeta: j \neq i} \min(D_j(t), x_j(t))[p_{ji}(t)]. \quad (3)$$

Since we do not know the number of trips that are going to be made on each origin-destination link beforehand, we have to use expectations in the bicycle-system models. The expected number of bicycle trips made during the modelling time is given by

$$\sum_{t=0}^K \sum_{i \in \zeta} \sum_{j \in \zeta: j \neq i} E(\min(D_i(t), x_i(t)) [p_{ij}(t)]). \quad (4)$$

Let us define the following variables

$$y_i(t) = E(x_i(t)) \quad (5)$$

$$y_{ij}(t) = E(\min(D_i(t), x_i(t)) [p_{ij}(t)]) \quad (6)$$

$$y_{ii}(t) = y_i(t) - \sum_{j \in \zeta: j \neq i} y_{ij}(t) \quad (7)$$

With these definitions we will describe some structural properties of $y_{ij}(t)$ by defining and proving three lemma's.

The first lemma defines the maximum of the expected number of bicycles moving from station i to station j during time period t .

Lemma 1. $y_{ij}(t) \leq r_{ij}(t)$

Proof. This lemma can be proven with the use of some definitions

$$y_{ij}(t) = E(\min(D_i(t), x_i(t)) [p_{ij}(t)]) \quad (8)$$

$$r_{ij}(t) = E(D_i(t) [p_{ij}(t)]) \quad (9)$$

We also know that

$$E(\min(D_i(t), x_i(t)) [p_{ij}(t)]) \leq E(D_i(t) [p_{ij}(t)]) \quad (10)$$

To see why this is true, lets say we have two random variable, U and O . If we know that $U(\omega) \leq O(\omega)$ for all events ω , then we can conclude that $E(U) \leq E(O)$. Now both sides of equation (10) are expectations of random variables and for every demand realization $d_i(t)$ it holds that $\min(d_i(t), x_i(t)) \leq d_i(t)$. Therefore we can conclude that equation (10) holds. This leads to the conclusion that $y_{ij}(t) \leq r_{ij}(t)$. \square

The next lemma describes a relationship between the expected number of bicycles at a station at the beginning of the next time period by using the expectations of this time period.

Lemma 2.

$$y_i(t+1) = y_i(t) - \sum_{j \in \zeta: j \neq i} y_{ij}(t) + \sum_{j \in \zeta: j \neq i} y_{ji}(t)$$

Proof. This lemma follows from the relation described in equation (3). If we take the expectation of all parts of that equation and combine them with the definitions of $y_i(t)$ and $y_{ij}(t)$, we get this lemma. This means that the expected number of bicycles at station i at the beginning of time period $t+1$ is equal to the expected number of bicycles at station i at the beginning of time period t minus the expected departures from station i during time period t plus the expected number of arrivals at station i during time period t . \square

The last lemma describes the relationship between the ratio of two expected number of trips from one station with different destinations and passenger arrival rates.

Lemma 3.

$$\frac{y_{ij}(t)}{y_{il}(t)} = \frac{r_{ij}(t)}{r_{il}(t)}$$

Proof. To proof this we will have to use some definitions

$$y_{ij}(t) = E(\min(D_i(t), x_i(t))[p_{ij}(t)]) = E\left(\sum_{k=1}^{\min(x_i(t), D_i(t))} Q_k(p_{ij}(t))\right) \quad (11)$$

Similarly,

$$y_{il}(t) = E(\min(D_i(t), x_i(t))[p_{il}(t)]) = E\left(\sum_{k=1}^{\min(x_i(t), D_i(t))} Q_k(p_{il}(t))\right) \quad (12)$$

Next we can use the fact that $E(Q_k(p_{ij}(t))) = p_{ij}(t)$, and $E(Q_k(p_{il}(t))) = p_{il}(t)$ as this follows the definition of the expectation of a Bernoulli sequence. We will combine these definitions and condition on $\min(x_i(t), D_i(t))$.

$$\begin{aligned} \frac{y_{ij}(t)}{y_{il}(t)} &= E\left(\frac{\sum_{k=1}^{\min(x_i(t), D_i(t))} Q_k(p_{ij}(t))}{\sum_{k=1}^{\min(x_i(t), D_i(t))} Q_k(p_{il}(t))}\right) = E\left(E\left(\frac{\sum_{k=1}^{\min(x_i(t), D_i(t))} Q_k(p_{ij}(t))}{\sum_{k=1}^{\min(x_i(t), D_i(t))} Q_k(p_{il}(t))} \mid \min(x_i(t), D_i(t))\right)\right) \\ &= E\left(E\left(\frac{\min(x_i(t), D_i(t))p_{ij}(t)}{\min(x_i(t), D_i(t))p_{il}(t)}\right)\right) = \frac{p_{ij}(t)}{p_{il}(t)} \end{aligned} \quad (13)$$

Now we will use the definitions of $p_{ij}(t)$ and $p_{il}(t)$.

$$\frac{y_{ij}(t)}{y_{il}(t)} = \frac{\frac{r_{ij}(t)}{\sum_{m \in \zeta; m \neq i} r_{im}(t)}}{\frac{r_{il}(t)}{\sum_{m \in \zeta; m \neq i} r_{im}(t)}} = \frac{r_{ij}(t)}{r_{il}(t)} \quad (14)$$

□

2.2 Stochastic Network Flow model

The first model Shu et al [1] constructs is a Stochastic Network Flow Model. This model maximizes the total amount of trips by determining the best initial allocations of the bicycles with a given bicycle utilization requirement. Some of the sets, parameters and variables are already described in section 2.1. Only the utilization requirement needs to be introduced in this section.

- **Parameter β** : the bicycle utilization requirement over the modelling time. This is equal to the average number of time periods each bicycle needs to be utilized.

The Stochastic Network Flow Model is now defined as follows.

$$Z^*(\beta) = \max_{x_i(0), y_{ij}(t)} \left(\sum_{t=0}^K \sum_{i \in \zeta} \sum_{j \in \zeta; j \neq i} y_{ij}(t) \right) \quad (15)$$

subject to

$$y_i(t+1) = y_i(t) - \sum_{j \in \zeta; j \rightarrow i} y_{ij}(t) + \sum_{j \in \zeta; j \leftarrow i} y_{ji}(t) \quad \forall i \in \zeta, t \in T \quad (16)$$

$$\sum_{t=0}^K \sum_{i \in \zeta} \sum_{j \in \zeta; j \neq i} y_{ij}(t) \geq \beta \sum_{i \in \zeta} x_i(0) \quad (17)$$

$$y_i(t) = y_{ii}(t) + \sum_{j \in \zeta; j \rightarrow i} y_{ij}(t) \quad \forall i \in \zeta, t \in T \quad (18)$$

$$\frac{y_{ij}(t)}{y_{il}(t)} = \frac{r_{ij}(t)}{r_{il}(t)} \quad \forall i, j, l \in \zeta \quad (19)$$

$$y_i(0) = x_i(0) \quad \forall i \in \zeta \quad (20)$$

$$0 \leq y_{ij}(t) \leq r_{ij}(t) \quad \forall i, j \in \zeta, i \neq j, t \in T \quad (21)$$

In this problem $Z^*(\beta)$ is the optimal objective value for a given β . In equation (15) the total number of trips made in the analysed time periods $1..K$ is maximized.

The first set of constraints in equation (16) states that the amount of bicycles at the beginning of time period $t + 1$ is equal to the amount of bicycles at the beginning of time period t minus the departures from station i during time period t plus the arrivals at station i during time period t . These constraints follow from lemma 1.

The second set of constraints in equation (17) makes sure that the bicycle utilization requirement β is met.

The third set of constraints in equation (18) states that the number of bicycles at station i at the beginning of time period t should always equal to the number of bicycles that will stay at station i during time period t plus the number of bicycles that will leave station i during time period t .

The fourth set of constraints in equation (19) follows from lemma 3. The fifth set of constraints in equation (20) state that the number of bicycles at station i at the initial time period should be equal to the number of bicycles assigned to station i during the initial allocation of the bicycles.

Then finally the last set of constraints in equation (21) make sure that the number of trips made between two stations can never be negative and can never become more than the customer arrival rate of trips between the same two stations during all time periods t . These constraints follow from lemma 2.

It is also possible to determine the optimal initial bicycle allocation for a given number of bicycles with the Stochastic Network Flow Model. Lets say that the total number of bicycles in the system is equal to N .

$$\sum_{i \in \zeta} x_i(0) = N \quad (22)$$

If we add equation (22) to the Stochastic Network Flow Model, it will determine the optimal initial bicycle allocation for a given number of bicycles in the system (N).

2.2.1 Upper bound

The Stochastic Network Flow Model described above is very effective in providing an estimation of the performance of a bicycle-sharing system. However, it remains an estimation which means that it is not exact. The optimal solution of the model can suppress certain trips if that generates a higher objective value. For example, take a three node bicycle-sharing system with two bicycles located at station 3. Now suppose $r_{31}(0) = r_{32}(0) = 1$, $r_{23}(t) = r_{32}(t) = 1$ for all $t > 1$, and $r_{ij}(t) = 0$ otherwise. If a bicycles moves from station 3 to station 1 during time period 0, then the bicycle will be stuck there for the remaining time periods. In real time and in the simulation this will happen eventually. However, the number of trips made over the whole day will be maximized if both bicycles move between station 2 and 3. The Stochastic Network Flow Model thus suppresses the bicycles from moving to station 1 in order to maximize the objective.

2.3 Equilibrium State Model

To determine which bicycle reallocation should take place, we have to know the equilibrium state of the bicycle-sharing system. As $t \rightarrow \infty$ we expect $y_i(t+1) = y_i(t)$ in the equilibrium state. If we let $y_{ij} = \lim_{t \rightarrow \infty} y_{ij}(t)$ and define N as the total number of bicycles in the system, then the following model can be used to determine the equilibrium state of the bicycle-sharing system.

$$Z^* = \max \sum_{i,j \in \zeta: j \neq i} y_{ij} \quad (23)$$

subject to

$$\sum_{j \in \zeta: j \neq i} y_{ij} = \sum_{j \in \zeta: j \neq i} y_{ji} \quad \forall i \in \zeta \quad (24)$$

$$\frac{y_{ij}}{y_{il}} = \frac{r_{ij}}{r_{il}} \quad \forall i, j, l \in \zeta \quad (25)$$

$$0 \leq y_{ij} \leq r_{ij} \quad \forall i, j \in \zeta \quad (26)$$

$$\sum_{i \in \zeta} (y_{ii} + \sum_{j \in \zeta: j \neq i} y_{ij}) = N \quad (27)$$

The objective in equation (23) maximizes the total number of trips made in the equilibrium state. The first set of constraints in equation (24) states that the number of trips arriving at station i during time period t should be equal to the number of bicycles departing from station i during time period t . The set of constraints in equations (25) and (26) are almost the same as the constraints in equation (19) and (20) respectively, except that these constraints do not depend on the time periods. The last constraint in equation (27) states that each of the N bicycles should either be used in a trip or must stay at the same station in the equilibrium state.

2.3.1 Adjustment of the solution

This model will determine the movements of the bicycles in the equilibrium state. It does not however show where the bicycles that are not used will be when time goes to infinity. Shu et al [1] suggests moving the bicycles to the sink node. These are the nodes with a larger inflow than outflow of bicycles. However, they do not mention how the bicycles should be divided over several sink nodes. In this report we will assume that these bicycles will 'sink' in the node with the largest inflow of bicycles compared to the outflow.

3 Numerical Experiments

In this section we will verify both models described in section 2 by comparing them to the results of a simulation. To do so we will firstly introduce the numerical example. Then we will describe the simulation and lastly we will compare some results of both models with the results of the simulation.

3.1 Numerical example

This numerical example consists of three nodes and arrival rates that are also used in Shu et al [1]. The customer arrival rates are assumed to be Poisson distributed. All nodes are connected and each node represents a station with docks for the bicycles. It is assumed that the duration of each trip is always shorter than one time period of fifteen minutes. However, if for example a bicycle from station 1 arrives at station 2 before the end of a time period, it is assumed that the bicycle can not be used until the start of the new time period. The arrival rates are time-invariant, which means that the customer arrival rates at each node will be the same for each time period.

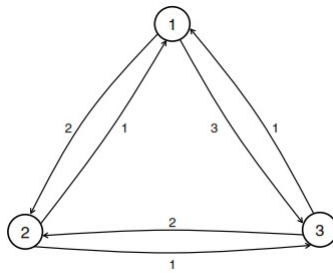


Figure 1: Three Node Example

Figure 1 displays the three node example with the customer arrival rates on the edges. For example, the customer arrival rate of station 1 who want to travel to station 2 is two.

3.2 CPLEX

Both models are implemented in IBM ILOG CPLEX Optimization Studio 12.6.1, or briefly referred to as CPLEX. This is a optimization software package that can be connected to several programming languages. For this report we imported CPLEX into Eclipse and use the Java programming language. Eclipse is a open source framework that allows you to install new plug-ins.

3.3 Simulation

To verify the models described in section 2 we used two forms of simulations. The first program we used is Anylogic. This program gives a nice visualization of the system and improves the intuition. The second is a simulation in Eclipse in Java language. The most important profit of this program and programming language is the fact that it allows to get numerical results in a short time period.

3.3.1 Anylogic

In this program, several simulation tools are optional. For this simulation an agent based simulation. In such simulations the agents can behave certain ways and they can transit between state. Persons can for example be infected, sick or healthy and they can transition between the three states. These states and their possible transition are visualised and connected in a state chart. The state chart of the numerical example is shown in figure 2.

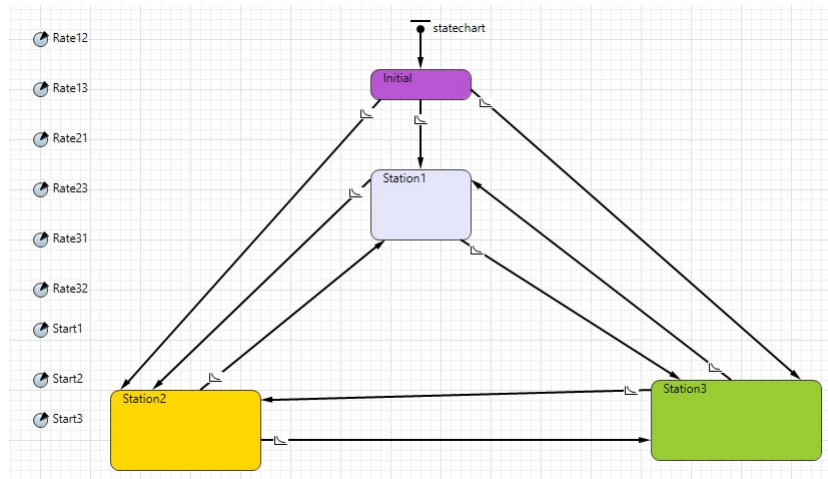


Figure 2: Anylogic state chart

In this simulation, the bicycles are the agents and the stations are their states. A bicycle can transit between the stations. The transitions are triggered by the arrivals of customers. The Poisson arrival process is build into Anylogic and we only have to define the rates. Now we can define variables and statistics that will keep track of what happens during the simulation. For example in this simulation a diagram showing the number of bicycles at each station appears on the screen.

When an agent enters a state, it can perform a entry action. Also when an agent leaves a state, it can perform a exit action. These actions can, for example, update variables or change the colour of the agent depicted on the screen. A screenshot of a simulation run of this example is shown in figure 3.

The bicycles (depicted as persons since the program does not have bicycles shapes) are changing colour during the simulation run, depending on in which state the bicycles are. During a simulation run the statistics and the diagram containing the number of bicycles at each station are automatically updated. If you set a model time as ending time of one simulation run, you can repeat the simulation run and generate results. You can even change parameters for each simulation, which means that you can see what happens when the parameters change. These are just some things that are possible using Anylogic but of course there are many more possibilities within this program.

3.3.2 Java

To describe this simulation, we need to define the following sets, parameters and variables that are used in the simulation.

- **Set ζ** : the set of all stations in the bicycle-sharing system.
- **Set L** : the set of number of simulations (1...H).
- **Set T** : the set of all K time periods (1...K).

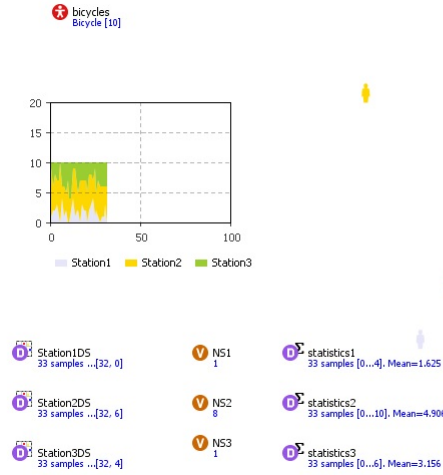


Figure 3: Screenshot of running Anylogic

- **Parameter** $r_{ij}(t)$: the customer arrival rate at station i with destination station j during time period t .
- **Variable** $c_{ij}(t)$: the number of trips made from station i to station j during time period t .
- **Variable** $b_i(t)$: the number of bicycles available at station i at the beginning of time period t .
- **Variable** m_j : the movements made from the current station i to station j during the current time period t .
- **Variable** M_{ij} : movements from station i to station j during current time period t .
- **Variable** $o(h)$: the total number of trips made during simulation run h
- **Variable** $u(h)$: the bicycle utilization during simulation run h .
- **Variable** $L_{ij}(h)$: the last bicycle movements of simulation run h .

With these definitions, we can now describe the main steps of the simulation. Some steps are later explained in more detail.

1. Initialization: $H = 2000, K = 50, o, u, L$
2. Read data, fill $r_{ij}(t)$ and $b_i(0)$
3. **For** all $h \in L$
4. Initialization: c, b
5. **For** all $t \in T$
6. Initialization M
7. **For** all $i \in \zeta$
8. **if** $b_i(t) \geq 0$
9. Initialization m
10. $minimum = \min(b_i(t), \sum_j r_{ij}(t))$
11. **For** $1 \dots minimum$
12. Determine destination this bicycle
13. Save the movement in m

14. **Next**
15. **End if**
16. Save movements of bicycles from station i in M
17. **Next**
18. Process all movements in $c_{ij}(t)$ and $b_i(t)$
19. **Next**
20. Calculate $o(h)$, $u(h)$ and $L_{ij}(h)$
21. **Next**

During step 2 in the overview above, the customer arrival rates and the initial bicycle distribution are imported into the program. The data is stored in text-files which are imported and read during this step. In this step we can also tell the program which customer arrival rates should be saved for which time periods. This allows us to simulate with time-varying customer arrival rates. Of course it is also possible to work with time-invariant customer arrival rates. In this case $r_{ij}(t)$ will be the same for all t . In step 4 the variables c and b are initialized. After each simulation run, these variables will be initialized in order to make them useful for the next simulation run.

The next initialization, the initialization of M , occurs in step 6. For each time period t of this simulation run h we have to create M because this variable will contain all the movements of this time period t . When we move to the next time period $t + 1$, this variable will be initialized again such that we can use them for the next time period $t + 1$.

Now that we are in simulation run h during time period t , we have to determine which movements will take place. This will be done by iterating through the stations and determining how many bicycles will leave each station to each destination. So now we will determine which customers will take a bicycle station i during time period t in simulation run h .

In step 8 we will firstly check whether there are bicycles available at station i before we start assigning customers to bicycles. If there are bicycles available, we will initialize m for this station i in which the number of bicycles that leave this station with destination j during this time period t is saved for all j . Next we decide in step 10 how many bicycles can leave this station. As explained in section 2 we have to take the minimum of the number of bicycles available and the sum of all customers arrival rates of station i .

Now that we know how many bicycles will leave station i , we have to decide what their destinations will be. This is done with a transformation of the Poisson arrival rates. For each bicycle that will be moved we will generate the transformation numbers f_j for each possible destination station j . We start by initializing a new random number generator and then draw a number v_j from the uniform distribution between 0.0 and 1.0 for each $j \in \zeta$ and $j \neq i$. With this v_j and $r_{ij}(t)$ we can now get to f_j with the following formula:

$$f_j = \frac{-\log(1.0 - v_j)}{r_{ij}(t)} \quad (28)$$

Next we decide which station got the lowest f_j as this will be the destination of this bicycle. Once we decided the destination of this bicycle, we will save this movement in m in step 12 and continue to the next bicycle.

We will do this for all bicycles that will leave this station and then save the movements from this station i in M in step 16. If we have decided what all movements will be from all stations $i \in \zeta$ during this time period, we will process these movements by updating $c_{ij}(t)$ and $b_i(t)$ in step 18.

When we arrive at step 20 we will determine how many trips are made during this simulation run and save that value for this simulation h in $o(h)$. We will also determine the bicycle utilization of this simulation h and save it in $u(h)$. Then we will save the last movements M_{ij} of this simulation run h in $L_{ij}(h)$.

These steps are repeated until all H simulation runs are completed. Now we can calculate the needed numbers with the help of $o(h)$ and $u(h)$. For the validation of the Stochastic Network Flow Model we will use the average of $o(h)$ and $u(h)$ to compare to the model results. For the Equilibrium State Model we will use the averages of the last movements of each simulation run by taking the average of $L_{ij}(h)$.

3.4 Comparison

In this section we will compare the results of the Stochastic Network Flow Model and the Equilibrium State Model with the results of the simulation.

3.4.1 Stochastic Network Flow Model

To verify the Stochastic Network Flow Model we first implemented the numerical example in CPLEX with constant arrival rates during 50 time periods. This model allows you to set a utilization requirement(β) as described in section 2.2. We solved the model for different utilization requirements. This model then determines the initial bicycle distribution such that the number of trips made will be maximized (note that this also means that the model determines the number of bicycles in the system). Next we used this bicycle distribution as input for the simulation. This simulation simulates the trips made during 50 time periods and calculates the total number of trips made and the bicycle utilization. This is repeated 2000 times and then we took the average of both the number of trips made and the bicycle utilization.

Table 1: Comparison of the Stochastic Network Flow Model and the simulation

<i>Deterministic</i>			<i>Simulation</i>	
% time periods a bicycle is used	Objective (number of trips)	Number of bicycles in the system	Average % time periods a bicycle is used	Average objective (number of trips)
0%	500.00	157.00	6.22%	487.97
10%	428.75	85.75	9.79%	419.83
20%	340.83	34.08	19.58%	333.58
30%	314.62	20.97	29.10%	305.08
40%	302.96	15.15	39.06%	295.85
50%	296.38	11.86	48.48%	287.48
60%	292.14	9.74	58.15%	283.18
70%	289.19	8.26	67.81%	280.05
80%	287.02	7.18	76.00%	272.84
90%	285.00	6.33	81.17%	256.90
100%	283.33	5.67	80.68%	228.74

The results of this comparison are shown in table 1. Firstly we look at the result of the deterministic model. An increase in the bicycle utilization requirement leads to a decrease in the objective. We also see a decrease in the optimal number of bicycles in the system. As a bicycle now needs to be used during more time periods, according to the bicycles utilization requirement, there can be less bicycles in the system. If less bicycles are available this will also decrease the total number of trips made as more customers are denied.

If we compare the bicycle utilization requirements in the first column of table 1 with the bicycle utilization in the simulation in the fourth column of table 1 we see that the model always overvalues the utilization. This can be explained by the fact that the model is an estimation and gives an upper bound of the number of trips made as described in section 2.2.1. We see that the difference between both values increases when the requirement increases. When we look at the objectives of the deterministic model and the simulation, we see the same pattern.

If we look at the last two rows of table 1 we see that the bicycle utilization in the simulation actually decreases. This suggests that the Stochastic Network Flow Model is not always capable of estimations with a bicycle requirement of 100%. The model suppresses more trips to satisfy the bicycle utilization requirement which leads to a bigger gap between the model and the simulation. However, the simulation will not suppress trips and therefore it could happen that the bicycle utilization actually decreases slightly.

3.4.2 Equilibrium State Model

For the comparison of this model with the simulation, we assumed that there are 10 bicycles in the system. The results of this model are stated in table 2

Table 2: Equilibrium State Model with numerical example

Origin \ Destination	1	2	3
1	0.00	0.67	1.00
2	1.00	4.33	1.00
3	0.67	1.33	0.00

Number of trips made in the equilibrium state

As we see in table 2 only 5.66 bicycles move in the equilibrium state while 4.33 are staying at station 2. When comparing the expected number of bicycles at station 2 to the customer arrival rates in figure 1, it can be seen that station 2 is a sink node. This means that bicycles will sink into this station when time increases. This station will have more bicycles than customers arriving. Each arriving customer can use a bicycle and the other bicycles will stay at the station.

The results of this model are compared with a simulation run of 50 time periods. The simulation ran 2000 times and after each simulation the last movements were saved. After that we took the average of these movements. The simulation needs an initial bicycle distribution as input in order to simulate the trips that the bicycles will make. This distribution can be chosen randomly but in this case we decided to use the same initial bicycle distribution as Shu et al [1]. This means that there are five bicycles at station 1, two bicycles at station 2 and, three bicycles at station 3 at the beginning of each simulation run. The result of these simulation are stated in table 3

Table 3: Simulation with the numerical example

Origin \ Destination	1	2	3
1	0.00	0.67	1.00
2	1.00	4.19	1.00
3	0.67	1.33	0.14

Average number of trips made during the last time period

Most values in table 3 are the same as in table 2. The main difference however, is the fact that some of the not moving bicycles stay at station 3 in the equilibrium state. When we look at the customers arrival rates in figure 1 you can see that station 3 also has a overflow of arriving bicycles compared to arriving customers. However, for station 2 this overflow is larger. So in some cases, there might be bicycles staying in station 3 during the last of the 50 time periods. We see, nonetheless, that this only happens on rare occasions, thus that in most cases the bicycles in station three will have moved to station 2 already.

In table 4 the number of bicycles at each station in the equilibrium state are stated for the model and the simulation.

Table 4: Equilibrium State Model compared to simulation

	Deterministic	Simulation	Abs. Diff.	% Diff. deterministic
Avg no. of bicycles at station 1	1.67	1.67	0.00	0.00%
Avg no. of bicycles at station 2	6.33	6.19	0.14	2.21%
Avg no. of bicycles at station 3	2.00	2.14	0.14	7.00%

To compare the deterministic model and the simulation we take their absolute difference as a percentage of the number of bicycles in the equilibrium state according to the deterministic model. From table 4 we can conclude that the pattern of the number of bicycles at each station in the equilibrium state is the same for the deterministic model and the simulation. However, on average there is still a difference of 3.07% per station.

4 Case studies

In this case study we will implement both models described in section 2 with data from the metro stations in Rio De Janeiro. Firstly we will describe how we use such data to estimate the customer arrival rates. Then we will describe the data we received and how we used it. Lastly we will look at the results of both models and the simulation.

4.1 Estimation arrival rates

For many cities a lot of data is available about people travelling. They know how many people use the bus, metro or train. However, in many cases there is no information available about how many people travel on some origin-destination route. Often we only know how many people get into a system at each station and we know how many people leave the system at each station. Often this data is also collected per time period.

For a good working system, we need to know how many people travel on a certain origin-destination route. If this is not known, then the origin-destination demand can be estimated using only the number of people getting into the system at each station and the number of people leaving the system at each station during each time period. These estimates can be used to determine the customer arrival rates of a certain origin-destination combination.

In the article by Marujo [3] a algorithm is developed to estimate these customer arrival rate. These estimations will then be implemented into the models and simulations as the $r_{ij}(t)$ parameter.

4.2 Rio De Janeiro

The data set we used in this case study, is estimated with the algorithm described in Marujo [3]. This data set contains the origin-destination estimated customer arrival rates for 35 metro stations in Rio De Janeiro. The names and codes we used for these station can be found in the Appendix. The data set contains estimations for each link on three different moments of a day, and an estimation of an average weekday. The three moments are the morning peak, lunch time and, the evening peak. Each customer arrival rates is measured over a 3 hour period. However, with the use of the Poisson Thinning lemma (see Ross [2]), we can use these rates for smaller time periods. In this case we decided to use time periods of half hours as all the trips should take no longer than one time period. The bicycle-sharing system in Rio De Janeiro possesses 600 bicycles in total.

4.3 Stochastic Network Flow Model

With this data from the metro stations in Rio De Janeiro, we would like to verify the Stochastic Network Flow Model with time varying customer arrival rates. We choose to look at one day and used the four different customer arrival rate sets described in section 4.2. The day starts with a morning peak from 06:30 till 09:30, the lunch break from 11:00 till 14:00 and, a evening peak from 17:00 till 20:00. The time in-between is filled with the regular weekdays data. All customer arrival rates were scaled to half hour periods. This means that there are 33 time periods in the model and a simulation run. A simulation run is repeated 2000 times and then the results were collected. The comparison values are shown in table 5.

Just like in the numerical experiment we see the same pattern when the bicycle utilization requirement increases. Also both the bicycle utilization and the number of trips made is higher in the model results than the simulation results. However, in this case we see the same values for a bicycle utilization requirement of 0%, 10% and, 20%. This means that the maximum number of trips possible with these customer arrival rates can already be achieved when there are 314235.18 bicycles in the system. Also the bicycle utilization will be at least 20% of the time periods. If we calculate the overall performance as the percentile difference between the bicycle utilization requirement, we will see that the model seemed to perform better for the numerical example. This model is not capable of capturing this case.

In the optimal objective without a bicycle utilization requirement, the bicycles will be used for some percentage of the time periods. If this percentage is higher than your bicycle utilization requirement, then the estimation will differ more than with requirements larger than this percentage.

Table 5: Comparison of the Stochastic Network Flow Model and the simulation with data from Rio De Janeiro

<i>Deterministic</i>			<i>Simulation</i>	
% time periods a bicycle is used	Objective (number of trips)	Number of bicycles in the system	Average % time periods a bicycle is used	Average objective (number of trips)
0%	2244744.93	314235.18	20.13%	208736.47
10%	2244744.93	314235.18	20.13%	208736.47
20%	2244744.93	314235.18	20.13%	2087837.91
30%	2077944.73	209893.41	27.94%	1934994.39
40%	1850759.35	140209.04	37.19%	1720781.37
50%	1598014.43	96849.36	46.52%	1486667.44
60%	982836.63	49638.21	56.79%	926791.44
70%	567563.19	24569.84	66.34%	537874.38
80%	381509.79	14451.13	76.18%	363269.53
90%	274260.24	9234.35	86.17%	262577.05
100%	96525.05	2925.00	96.53%	93177.45

4.4 Equilibrium State Model

We also used the data sets described in section 4.2 in the Equilibrium State Model and again compared the results with the results of a simulation. We did this for all four data sets and the whole comparison tables can be found in the appendix. A simulation run was again 50 time periods and was repeated 2000 times. In table 6 the average differences between the model and the simulation of the number of bicycles at each station in the equilibrium state are shown.

Table 6: Average difference between the Equilibrium State Model and the simulation

Data Set	average % difference
<i>Weekday</i>	0.56%
<i>Morning</i>	0.88%
<i>Lunch</i>	0.58%
<i>Evening</i>	0.64%

We see that these numbers are rather smaller than the differences in the numerical experiment. These data sets contain more stations and also more bicycles. So more stations and bicycles in the system will lead to a higher accuracy of the Equilibrium State Model.

5 Conclusion

The Equilibrium State Model seems to get close to bicycles distribution of the moving bicycles when t goes to infinity. However, it does not take into account the distribution of the bicycles that do not move in the equilibrium state.

From the comparison of the Equilibrium State Model with the simulation we can conclude that the model tempts to be closer to the simulated values when more bicycles are available and when there are more stations in the system.

With the Stochastic Network Flow Model the number of bicycles needed to obey a utilization requirement can be determined. However, it does not seem to be capable of estimating all cases. In the optimal objective without a bicycle utilization requirement, the bicycles will be used for some percentage of the time periods. If this percentage is higher than your bicycle utilization requirement, then the estimation will differ more than with requirements larger than this percentage.

6 Discussion

First of all we would like to mention that the output using the data sets from Rio De Janeiro should be downscaled. The data set represents the arrival rate of customer who want to travel with the metro. It is not realistic to assume that all metro passengers will choose to use the bicycles if they are distributed correctly.

In future research we would firstly like to extend the Anylogic model such that it can handle more nodes in a system. It would also be a good step to make a moving picture of a bigger simulation to see where the bicycles are moving. This could give some insight into improvements of the system. Secondly we saw that Shu et al [1] did not mention how the bicycles that are not moving should be distributed over the stations in the equilibrium state. Since most of these bicycles will 'sink' into one or more stations, we think it would be interesting to see which stations they 'sink' into with what numbers as this can give us insight into which bicycle redistribution should take place if we want to redistribute the bicycles after one day for example. And lastly we would like to analyse the bicycle redistribution model constructed by Shu et al [1] because we think redistribution of the bicycles is the main reason for looking into a Equilibrium State Model. Redistribution can also make the whole system more profitable and trustworthy which are both important for the reputation of bicycle-sharing systems.

References

- [1] Shu, J., Chou, M., Liu, Q., Teo, C.-P., Wang, I.-L., (2013). Models for effective deployment and redistribution of bicycles within public bicycle-sharing systems. *Operations Research. Vol. 61, No.6, pp. 1346-1359.*
- [2] Ross, S.M., (2014). Introduction to probability models 11th edition.
- [3] Marujo, L.G., (2015). OD matrix estimation and the mode choice function.

7 Appendix

7.1 Stations in Rio De Janeiro

This table contains all the stations we used in this report with the codes used in the tables.

Table 7: Station names with codes

Name Station			Code Station
Ipanema/General	Osório		IGO
Cantagalo			CTG
Siqueira	Campos		SCP
Cardeal	Arcoverde		CAV
Botafogo			BTF
Flamengo			FLA
Largo	do	Machado	LMC
Catete			CTT
Glória			GLR
Cinelândia			CNL
Carioca			CRC
Uruguaiana			URG
Presidente	Vargas		PVG
Central			CTR
Praça	Onze		POZ
Estácio			ESA
Afonso	Pena		AFP
São	Francisco	Xavier	SFX
Saens	Peña		SPN
Cidade	Nova		CNV
São	Cristóvão		SCR
Maracanã			MRC
Triagem			TRG
Maria	da	Graça	MGR
Nova	América/Del	Castilho	DCT
Inhaúma			INH
Engenho	da	Rainha	ERN
Thomaz	Coelho		TCL
Vicente	de	Carvalho	VCV
Irajá			IRJ
Colégio			CLG
Coelho	Neto		CNT
Acari/Fazenda	Botafogo		AFB
Engenheiro	Rubens	Paiva	ERP
Pavuna			PVN

7.2 Equilibrium State Model - Simulation

The next table is a comparison of the Equilibrium State Model with the simulation with the weekday data of Rio De Janeiro. For each station the average amount of bicycles at that station is shown.

Table 8: Equilibrium State Model and Simulation with weekday data in Rio De Janeiro

	Deterministic	Simulation	Absolute Difference	%Difference from deterministic
SPN	41.36	41.47	0.11	0.27
SFX	7.10	7.06	0.04	0.56
AFP	7.00	6.93	0.07	1.00
ESA	7.81	7.86	0.05	0.64
POZ	4.39	4.4	0.01	0.23
CTR	79.54	79.72	0.18	0.23
PVG	8.12	8.2	0.08	0.99
URG	38.92	39.14	0.22	0.57
CRC	55.24	54.92	0.32	0.58
CNL	31.74	31.68	0.06	0.19
GLR	9.02	9	0.02	0.22
CTT	10.57	10.59	0.02	0.19
LMC	22.23	22.3	0.07	0.31
FLA	15.01	14.87	0.14	0.93
BTF	58.15	58.18	0.03	0.05
CAV	10.11	10.14	0.03	0.30
SCP	17.90	17.8	0.1	0.56
CTG	10.77	10.73	0.04	0.37
IGO	25.27	25.12	0.15	0.59
CNV	4.06	4.09	0.03	0.74
SCR	8.02	8.05	0.03	0.37
MRC	4.84	4.81	0.03	0.62
TRG	4.06	4.05	0.01	0.25
MGR	6.62	6.63	0.01	0.15
DCT	14.08	14.18	0.1	0.71
INH	5.03	5.06	0.03	0.60
ERN	4.17	4.13	0.04	0.96
TCL	2.26	2.23	0.03	1.33
VCV	12.78	12.84	0.06	0.47
IRJ	10.16	10.23	0.07	0.69
CLG	4.98	4.98	1.78E-15	0.00
CNT	8.95	9	0.05	0.56
AFB	2.41	2.43	0.02	0.83
ERP	2.35	2.4	0.05	2.13
PVN	45.05	45.2	0.15	0.33

The next table is a comparison of the Equilibrium State Model with the simulation with the morning data of Rio De Janeiro. For each station the average amount of bicycles at that station is shown.

Table 9: Equilibrium State Model and Simulation with morning data in Rio De Janeiro

	Deterministic	Simulation	Absolute Difference	%Difference from deterministic
SPN	33.48	33.41	0.07	0.21
SFX	7.00	7.11	0.11	1.57
AFP	7.30	7.2	0.1	1.37
ESA	11.02	11.08	0.06	0.54
POZ	10.53	10.59	0.06	0.57
CTR	46.88	47.03	0.15	0.32
PVG	13.54	13.64	0.1	0.74
URG	53.44	53.32	0.12	0.22
CRC	82.67	82.7	0.03	0.04
CNL	44.65	44.52	0.13	0.29
GLR	15.93	15.86	0.07	0.44
CTT	12.66	12.73	0.07	0.55
LMC	28.59	28.5	0.09	0.31
FLA	18.59	18.51	0.08	0.43
BTF	66.61	66.66	0.05	0.08
CAV	9.76	9.73	0.03	0.31
SCP	20.08	20.21	0.13	0.65
CTG	11.76	11.92	0.16	1.36
IGO	34.98	34.92	0.06	0.17
CNV	4.24	4.25	0.01	0.24
SCR	8.92	9.05	0.13	1.46
MRC	4.23	4.2	0.03	0.71
TRG	3.32	3.32	8.88178E-16	0.00
MGR	3.50	3.51	0.01	0.29
DCT	8.79	8.9	0.11	1.25
INH	1.61	1.63	0.02	1.24
ERN	1.52	1.5	0.02	1.32
TCL	0.67	0.69	0.02	2.99
VCV	4.84	4.84	8.88178E-16	0.00
IRJ	4.79	4.87	0.08	1.67
CLG	2.04	2.01	0.03	1.47
CNT	3.08	3.1	0.02	0.65
AFB	0.89	0.91	0.02	2.25
ERP	0.88	0.84	0.04	4.55
PVN	16.96	17.05	0.09	0.53

The next table is a comparison of the Equilibrium State Model with the simulation with the lunch data of Rio De Janeiro. For each station the average amount of bicycles at that station is shown.

Table 10: Equilibrium State Model and Simulation with lunch data in Rio De Janeiro

	Deterministic	Simulation	Absolute Difference	%Difference from deterministic
SPN	42.95	42.95	7.10543E-15	0.00
SFX	7.97	7.87	0.1	1.25
AFP	7.24	7.27	0.03	0.41
ESA	12.80	12.83	0.03	0.23
POZ	6.10	6.13	0.03	0.49
CTR	59.25	59.32	0.07	0.12
PVG	12.70	12.79	0.09	0.71
URG	52.03	52.34	0.31	0.60
CRC	66.39	66.12	0.27	0.41
CNL	38.87	39.07	0.2	0.51
GLR	10.10	10.17	0.07	0.69
CTT	11.41	11.28	0.13	1.14
LMC	24.96	25	0.04	0.16
FLA	13.84	13.82	0.02	0.14
BTF	55.12	55.06	0.06	0.11
CAV	12.04	12.1	0.06	0.50
SCP	19.77	19.64	0.13	0.66
CTG	11.17	11.2	0.03	0.27
IGO	26.19	26.06	0.13	0.50
CNV	5.96	5.95	0.01	0.17
SCR	8.52	8.48	0.04	0.47
MRC	5.48	5.49	0.01	0.18
TRG	2.85	2.82	0.03	1.05
MGR	4.81	4.81	1.77636E-15	0.00
DCT	12.71	12.82	0.11	0.87
INH	3.16	3.19	0.03	0.95
ERN	2.99	2.93	0.06	2.01
TCL	1.74	1.71	0.03	1.72
VCV	9.31	9.38	0.07	0.75
IRJ	7.51	7.47	0.04	0.53
CLG	3.61	3.58	0.03	0.83
CNT	5.42	5.41	0.01	0.18
AFB	1.62	1.63	0.01	0.62
ERP	0.98	0.99	0.01	1.02
PVN	32.50	32.49	0.01	0.03

The next table is a comparison of the Equilibrium State Model with the simulation with the evening data of Rio De Janeiro. For each station the average amount of bicycles at that station is shown.

Table 11: Equilibrium State Model and Simulation with evening data in Rio De Janeiro

	Deterministic	Simulation	Absolute Difference	%Difference from deterministic
SPN	39.86	39.81	0.05	0.13
SFX	5.86	5.89	0.03	0.51
AFP	7.02	7.12	0.10	1.42
ESA	5.83	5.86	0.03	0.51
POZ	3.51	3.59	0.08	2.28
CTR	85.38	85.16	0.22	0.26
PVG	4.54	4.61	0.07	1.54
URG	19.84	19.76	0.08	0.40
CRC	30.62	30.66	0.04	0.13
CNL	19.46	19.28	0.18	0.92
GLR	8.21	8.25	0.04	0.49
CTT	9.11	9.08	0.03	0.33
LMC	15.15	15.14	0.01	0.07
FLA	12.41	12.5	0.09	0.73
BTF	47.28	47.44	0.16	0.34
CAV	7.28	7.4	0.12	1.65
SCP	13.50	13.54	0.04	0.30
CTG	8.31	8.33	0.02	0.24
IGO	17.18	17.29	0.11	0.64
CNV	3.83	3.75	0.08	2.09
SCR	13.91	13.87	0.04	0.29
MRC	8.33	8.36	0.03	0.36
TRG	5.23	5.24	0.01	0.19
MGR	11.02	11.06	0.04	0.36
DCT	26.15	26.19	0.04	0.15
INH	8.44	8.33	0.11	1.30
ERN	7.20	7.23	0.03	0.42
TCL	4.96	4.95	0.01	0.20
VCV	20.92	20.81	0.11	0.53
IRJ	16.66	16.54	0.12	0.72
CLG	8.60	8.67	0.07	0.81
CNT	16.85	16.82	0.03	0.18
AFB	5.13	5.15	0.02	0.39
ERP	5.20	5.27	0.07	1.35
PVN	77.25	77.17	0.08	0.10

7.3 Appendix: Java Codes

7.3.1 Simulation with output for the Stochastic Network Flow Model

Listing 1: Simulation Stochastic Network Flow Model with input data Rio De Janeiro

```

1
2 import java.io.BufferedReader;
3 import java.io.BufferedWriter;
4 import java.io.FileNotFoundException;
5 import java.io.FileReader;
6 import java.io.FileWriter;
7 import java.io.IOException;
8 import java.util.Random;
9
10

```

```

11 public class MainSimulationRio {
12
13     public static void main(String[] args) throws FileNotFoundException, IOException {
14         int nd=1; //Denote number of days
15         int np=33; //Denote number of periods on a day
16         int NT=nd*np;
17         double NTD = (double) NT;
18         int H=2000; // Number of simulation runs
19         //int NB = 600; // Number of bicycles
20         int m=35; //number of nodes in the system
21         Double [][] rm = DataSet("ODMetroRioMorningt.txt", m);
22         Double [][] rl = DataSet("ODMetroRioLuncht.txt", m);
23         Double [][] re = DataSet("ODMetroRioEveningt.txt", m);
24         Double [][] rw = DataSet("ODMetroRioWeekdayt.txt", m);
25         Double[] Util = new Double[H];
26         Double[] Obj = new Double[H];
27         Double [][] x = new Double[m][NT]; //Create matrix that will hold all number of
            bicycles at each station
28         Double [][][] yj = new Double[m][m][H];
29         Double [][][] r = new Double[m][m][NT];
30         //Distribute all bicycles at the initialization
31
32         // Fill r
33         for (int i=0; i<m; i++){
34             for(int j=0; j<m; j++){
35                 for(int d=0; d<nd ; d++){
36                     for (int t= 33*d; t<(33*d)+6; t++){
37                         r[i][j][t]=(1.0/6.0)*rm[i][j]; // Fills the morning part on each day
38                     }
39                     for (int t= (33*d)+6; t<(33*d)+9; t++){
40                         r[i][j][t]=(3.0/28.0)*rw[i][j]; // Fills part between morning and
                            lunch on each day
41                     }
42                     for (int t= (33*d)+9; t<(33*d)+15; t++){
43                         r[i][j][t]=(1.0/6.0)*rl[i][j]; // Fills lunch part on each day
44                     }
45                     for (int t= (33*d)+15; t<(33*d)+21; t++){
46                         r[i][j][t]=(6.0/28.0)*rw[i][j]; // Fills part between lunch and
                            evening on each day
47                     }
48                     for (int t= (33*d)+21; t<(33*d)+27; t++){
49                         r[i][j][t]=(1.0/6.0)*re[i][j]; // Fills evening part on each day
50                     }
51                     for (int t= (33*d)+27; t<(33*d)+33; t++){
52                         r[i][j][t]=(6.0/28.0)*rw[i][j]; // Fills part after evening on each
                            day
53                     }
54                 }
55             }
56         }
57         Double[] xSNF = new Double[m];
58         for (int run=0; run<10; run++){
59
60             if (run==0){
61                 xSNF = readX("disxR33.0.txt", m);
62             }
63             else if (run==1){
64                 xSNF = readX("disxR29.7.txt", m);
65             }
66             else if (run==2){
67                 xSNF = readX("disxR26.4.txt", m);
68             }
69             else if (run==3){
70                 xSNF = readX("disxR23.1.txt", m);
71             }
72             else if (run==4){
73                 xSNF = readX("disxR19.8.txt", m);
74             }
75             else if (run==5){
76                 xSNF = readX("disxR16.5.txt", m);
77             }
78             else if (run==6){
79                 xSNF = readX("disxR13.2.txt", m);
80             }
81             else if (run==7){
82                 xSNF = readX("disxR9.9.txt", m);
83             }
84             else if (run==8){
85                 xSNF = readX("disxR9.9.txt", m);
86             }
87             else if (run==9){
88                 xSNF = readX("disxR9.9.txt", m);
89             }
90         }
91     }
92 }

```

```

79         xSNF = readX("disxR6.6.txt",m);
80     }else{
81         xSNF = readX("disxR3.3.txt",m);
82     }
83
84     double value= getSum(xSNF);
85     int NB = (int) value;
86     for (int i=0;i<m;i++){
87         x[i][0] = xSNF[i] ;
88     }
89
90     // Execute the simulation H times
91     for (int h=0;h<H;h++){
92         Double [][][] yh = new Double[m][m][NT];
93         yh=Simulation(NT,m,NB,r,x);
94         // Save the objective
95         Double [][] tempO1 = new Double[m][m];
96         Double [] tempO2 = new Double[m];
97         for(int i =0; i<m; i++){
98             for (int j =0; j<m ; j++){
99                 yj[i][j][h] = yh[i][j][NT-1]; //Save all last movements
100                if(i!=j){
101                    tempO1[i][j] = getSum(yh[i][j]); // summation over t
102                }else if (i==j){
103                    tempO1[i][j]=0.0;
104                }
105            }
106            tempO2[i] = getSum(tempO1[i]); // summation over t and over j
107        }
108        Obj[h] = getSum(tempO2); // summation over t , i and j
109
110        // Save the utilization percentage
111        Util[h] = (Obj[h] / NB)/NTD;
112    }
113
114    //Get the numbers, average of each 2000
115    Double [][] ya = new Double[m][m];
116    for (int i=0; i<m;i++){
117        for (int j =0;j<m;j++){
118            Double [] temp = new Double[H];
119            for (int h=0;h<H;h++){
120                temp[h] = yj[i][j][h];
121            }
122            ya[i][j] = round(getAverage(temp),2);
123        }
124    }
125
126    writeMatrix("SimRioWeekBeta.txt", ya, m);
127
128    System.out.println(" Utilization: "+ getAverage(Util));
129    System.out.println(" Objective: "+ getAverage(Obj));
130 }
131 }
132
133 public static Double [][][] Simulation (int NT, int m,int NB, Double [][][] r,Double
134 [][] x) throws FileNotFoundException, IOException{
135     //Initialization
136     Double [][][] y = new Double[m][m][NT]; //Create the matrix that will hold all
137     movements
138     //Fill x and y with 0.0
139     for(int i =0;i<m;i++){
140         for (int t=0;t<NT;t++){
141             if (t!=0){
142                 x[i][t]=0.0;
143             }
144             for (int j=0;j<m;j++){
145                 y[i][j][t]=0.0;
146             }
147         }
148     }

```

```

149 // Enter the for-loop of the actual simulation
150 for (int t=0;t<NT-1;t++){
151     Double [][] allMove = new Double[m][m];
152     for (int i=0;i<m;i++){
153         for (int j=0;j<m;j++){
154             allMove[i][j]=0.0;
155         }
156     }
157 }
158 for (int i=0;i<m;i++){
159     if (x[i][t]!=0.0){
160         x[i][t+1]=x[i][t]; // First get the previous value before adjusting it
161         Double [] mi= new Double[m];
162         mi=move(i,m,t,r,y,x); //What is the movement
163         allMove[i]=mi; //Save the movements
164         for (int a=0;a<m;a++){
165             if (a==i){
166                 y[i][a][t+1]= x[i][t] -getSum(mi); // Determine the number of
167                 bicycles that stayed at the station
168             }else{
169                 //We can already save the departures. The arrivals need to be
170                 saved after all movements are determined
171                 x[i][t+1] = x[i][t+1] - mi[a];
172                 y[i][a][t+1] = mi[a];
173             }
174         }
175     }
176     for (int i=0;i<m;i++){
177         for (int j=0;j<m;j++){
178             x[i][t+1] = x[i][t+1] + allMove[j][i];
179         }
180     }
181     return y;
182 }
183
184 public static double getSum(Double [] x){
185     double som=0.0;
186     for (int i=0;i<x.length;i++){
187         som = som + x[i];
188     }
189     return som;
190 }
191
192 public static double getAverage(Double [] x){
193     double average=0.0;
194     double sum=0.0;
195     for (int i=0;i<x.length;i++){
196         sum=sum+x[i];
197     }
198     average = sum/x.length;
199     return average;
200 }
201
202 public static Double [][] DataSet(String filename , int size) throws
203     FileNotFoundException , IOException {
204     String line = "" ;
205
206     Double [][] data = new Double[size][size];
207
208     FileReader fr= new FileReader(filename);
209     BufferedReader br = new BufferedReader(fr);
210     line=br.readLine();
211
212     String [] fline=line.split(";");
213     int c=fline.length;
214
215     for (int j =0; j<size; j++){
216         String [] theline=line.split(";");
217

```

```

218     Double [] td=new Double[ size ];
219     for (int i=0; i<c; i++){
220         td[i] = (double) ( Integer.parseInt(theline[i]));
221     }
222     data[j]=td;
223     if(j!=size){
224         line=br.readLine();
225     }
226 }
227
228
229 br.close();
230
231
232 return data;
233 }
234
235
236 public static Double[] readX(String filename, int size)throws FileNotFoundException
    , IOException{//Creating r
237     Double[] data = new Double[size]; //Create new array for initial distribution
238     String line = "";
239
240     FileReader fr= new FileReader(filename);
241     BufferedReader br = new BufferedReader(fr);
242     line=br.readLine();
243
244     String [] theline=line.split(",");
245     for (int i=0; i<size; i++){
246         data[i]= Double.parseDouble(theline[i]);
247     }
248     br.close();
249     return data;
250 }
251
252
253 public static Double[] move(int s, int m, int t, Double[][][] r,Double[][][] y,
    Double[][] x){
254     Double[] movement=new Double[m];
255     // Fill movement with 0.0
256     for(int i=0;i<m;i++){
257         movement[i]=0.0;
258     }
259
260     //Determine the total departure rate
261     int sumR=0;
262
263     for (int j =0;j<m;j++){
264         if (j!=s){
265             double thisrate= r[s][j][t];
266             sumR=sumR+ (int) thisrate;
267         }
268     }
269     //Decide how many bicycles can be moved
270     int minimum=0;
271     if (x[s][t]<= sumR){
272         double value=x[s][t];
273         minimum= (int) value;
274     }else{
275         minimum=sumR;
276     }
277
278     for (int k=0;k<minimum;k++){
279         Double[] pois = new Double[m];
280         for (int j =0;j<m;j++){
281             if(j!=s){
282                 Random rand = new Random();
283                 pois[j] = -Math.log(1.0 - rand.nextDouble()) / r[s][j][t];
284             }else{
285                 pois[j] = 10000000.0;
286             }
287         }

```

```

288     int minPlace=0;
289     double minValue=10000000.0;
290     for (int h=0;h<m;h++){
291         if (pois[h]<minValue){
292             minPlace=h;
293             minValue=pois[h];
294         }else if (pois[h]==minValue){
295             Integer [] ran=new Integer[2];
296             ran[0]=h;
297             ran[1]=minPlace;
298             minPlace=getRandom(ran);
299         }
300     }
301     movement[minPlace] = movement[minPlace] +1.0;
302 }
303
304 return movement;
305 }
306
307 public static int getRandom(Integer[] array) {
308     int rnd = new Random().nextInt(array.length);
309     return array[rnd];
310 }
311
312 public static void writeMatrix(String filename, Double [][] yp, int m){
313
314     String fileName = filename;
315
316     try {
317         // New Filewriter
318         FileWriter fileWriter = new FileWriter(fileName);
319         BufferedWriter bufferedWriter =new BufferedWriter(fileWriter);
320
321         for (int i=0;i<m;i++){
322             for (int j=0; j<m; j++){
323                 bufferedWriter.write(String.valueOf(yp[i][j]));
324                 bufferedWriter.write(" ");
325             }
326             bufferedWriter.newLine();
327         }
328
329         // Close files
330         bufferedWriter.close();
331     }
332     catch(IOException ex) {
333         System.out.println(
334             "Error writing to file "
335             + fileName + " ");
336     }
337 }
338
339 public static double round(double value, int places) {
340     if (places < 0) throw new IllegalArgumentException();
341
342     long factor = (long) Math.pow(10, places);
343     value = value * factor;
344     long tmp = Math.round(value);
345     return (double) tmp / factor;
346 }
347
348 }

```

7.4 Simulation without for the Equilibrium State Model

Listing 2: Simulation Equilibrium State Model with input data Rio De Janeiro

```

1
2
3 import java.io.BufferedReader;
4 import java.io.BufferedWriter;

```



```

5 import java.io.FileNotFoundException;
6 import java.io.FileReader;
7 import java.io.FileWriter;
8 import java.io.IOException;
9 import java.util.Random;
10
11
12 public class MainSimulationRioES {
13
14     public static void main(String[] args) throws FileNotFoundException, IOException {
15         int nd=1; //Denote number of days
16         int np=50; //Denote number of periods on a day
17         int NT=nd*np;
18         double NTD = (double) NT;
19         int H=2000; // Number of simulation runs
20         int NB = 600; // Number of bicycles
21         int m=35; //number of nodes in the system
22         Double [][] rm = DataSet("ODMetroRioEveningt.txt", m);
23         Double [] Util = new Double[H];
24         Double [] Obj = new Double[H];
25         Double [][] x = new Double[m][NT]; //Create matrix that will hold all number of
                bicycles at each station
26         Double [][][] yj = new Double[m][m][H];
27         Double [][][] r = new Double[m][m][NT];
28         //Distribute all bicycles at the initialization
29
30         // Fill r
31         for (int i=0;i<m;i++){
32             x[i][0] = 0.0;
33             for(int j=0;j<m;j++){
34                 for (int t=0; t<NT; t++){
35                     r[i][j][t] = rm[i][j];
36                 }
37             }
38         }
39
40         //Randomly distribute bicycles over the stations
41         for(int nb=0;nb<NB;nb++){
42             Random rand = new Random();
43             int place=rand.nextInt(m);
44             x[place][0] = x[place][0] + 1.0;
45         }
46
47
48         // Execute the simulation H times
49         for (int h=0;h<H;h++){
50             Double [][][] yh = new Double[m][m][NT];
51             yh=Simulation(NT,m,NB,r,x);
52             // Save the objective
53             Double [][] tempO1 = new Double[m][m];
54             Double [] tempO2 = new Double[m];
55             for(int i =0; i<m; i++){
56                 for (int j =0; j<m ; j++){
57                     yj[i][j][h] = yh[i][j][NT-1]; //Save all last movements
58                     if(i!=j){
59                         tempO1[i][j] = getSum(yh[i][j]); // summation over t
60                     }else if (i==j){
61                         tempO1[i][j]=0.0;
62                     }
63                 }
64                 tempO2[i] = getSum(tempO1[i]); // summation over t and over j
65             }
66             Obj[h] = getSum(tempO2); // summation over t , i and j
67
68             // Save the utilization percentage
69             Util[h] = (Obj[h] / NB)/NTD;
70         }
71
72         //Get the numbers, average of each 2000
73         Double [][] ya = new Double[m][m];
74         for (int i=0; i<m; i++){
75             for (int j =0;j<m;j++){

```

```

76         Double[] temp = new Double[H];
77         for (int h=0;h<H;h++){
78             temp[h] = yj[i][j][h];
79         }
80         ya[i][j] = round(getAverage(temp),2);
81     }
82 }
83
84 writeMatrix("SimRioWeekBeta.txt", ya, m);
85
86 System.out.println("Utilization: "+ getAverage(Util));
87 System.out.println("Objective: "+ getAverage(Obj));
88 }
89
90
91 public static Double[][][] Simulation (int NT, int m,int NB, Double[][][] r,Double
    [][] x) throws FileNotFoundException, IOException{
92     //Initialization
93     Double[][][] y = new Double[m][m][NT]; //Create the matrix that will hold all
        movements
94     //Fill x and y with 0.0
95     for (int i =0;i<m;i++){
96         for (int t=0;t<NT;t++){
97             if (t!=0){
98                 x[i][t]=0.0;
99             }
100            for (int j=0;j<m;j++){
101                y[i][j][t]=0.0;
102            }
103        }
104    }
105
106
107
108    // Enter the for-loop of the actual simulation
109    for (int t=0;t<NT-1;t++){
110        Double[][] allMove = new Double[m][m];
111        for (int i=0;i<m;i++){
112            for (int j=0; j<m ; j++){
113                allMove[i][j]=0.0;
114            }
115        }
116        for (int i=0;i<m;i++){
117            if (x[i][t]!=0.0) {
118                x[i][t+1]=x[i][t]; // First get the previous value before adjusting it
119                Double[] mi= new Double[m];
120                mi=move(i,m,t,r,y,x); //What is the movement
121                allMove[i]=mi; //Save the movements
122                for (int a=0;a<m;a++){
123                    if (a==i){
124                        y[i][a][t+1]= x[i][t] -getSum(mi); // Determine the number of
                            bicycles that stayed at the station
125                    }else{
126                        //We can already save the departures. The arrivals need to be
                            saved after all movements are determined
127                        x[i][t+1] = x[i][t+1] - mi[a];
128                        y[i][a][t+1] = mi[a];
129                    }
130                }
131            }
132        }
133        for (int i=0;i<m;i++){
134            for (int j=0;j<m;j++){
135                x[i][t+1] = x[i][t+1] + allMove[j][i];
136            }
137        }
138    }
139    return y;
140 }
141
142 public static double getSum(Double[] x){
143     double som=0.0;

```

```

144     for (int i=0;i<x.length;i++){
145         som = som + x[i];
146     }
147     return som;
148 }
149
150 public static double getAverage(Double[] x){
151     double average=0.0;
152     double sum=0.0;
153     for(int i=0;i<x.length;i++){
154         sum=sum+x[i];
155     }
156     average = sum/x.length;
157     return average;
158 }
159
160 public static Double[][] DataSet(String filename, int size) throws
    FileNotFoundException, IOException {
161     String line = "";
162
163
164     Double[][] data = new Double[size][size];
165
166     FileReader fr= new FileReader(filename);
167     BufferedReader br = new BufferedReader(fr);
168     line=br.readLine();
169
170     String[] fline=line.split(";");
171     int c=fline.length;
172
173
174     for(int j =0; j<size; j++){
175         String[] theline=line.split(";");
176         Double[] td=new Double[size];
177         for (int i=0; i<c; i++){
178             td[i] = (double) ( Integer.parseInt(theline[i]));
179         }
180         data[j]=td;
181         if(j!=size){
182             line=br.readLine();
183         }
184     }
185
186
187     br.close();
188
189
190     return data;
191 }
192
193
194 public static Double[] readX(String filename, int size) throws FileNotFoundException
    , IOException{//Creating r
195     Double[] data = new Double[size]; //Create new array for initial distribution
196     String line = "";
197
198     FileReader fr= new FileReader(filename);
199     BufferedReader br = new BufferedReader(fr);
200     line=br.readLine();
201
202     String[] theline=line.split(";");
203     for (int i=0; i<size; i++){
204         data[i]= Double.parseDouble(theline[i]);
205     }
206     br.close();
207     return data;
208 }
209
210
211 public static Double[] move(int s, int m, int t, Double[][][] r,Double[][][] y,
212     Double[][] x){
    Double[] movement=new Double[m];

```

```

213 // Fill movement with 0.0
214 for (int i=0;i<m;i++){
215     movement[i]=0.0;
216 }
217
218 //Determine the total departure rate
219 int sumR=0;
220
221 for (int j =0;j<m;j++){
222     if (j!=s){
223         double thisrate= r[s][j][t];
224         sumR=sumR+ (int) thisrate;
225     }
226 }
227 //Decide how many bicycles can be moved
228 int minimum=0;
229 if (x[s][t]<= sumR){
230     double value=x[s][t];
231     minimum= (int) value;
232 }else{
233     minimum=sumR;
234 }
235
236 for (int k=0;k<minimum;k++){
237     Double [] pois = new Double[m];
238     for (int j =0;j<m;j++){
239         if (j!=s){
240             Random rand = new Random();
241             pois[j] = -Math.log(1.0 - rand.nextDouble()) / r[s][j][t];
242         }else{
243             pois[j] = 10000000.0;
244         }
245     }
246     int minPlace=0;
247     double minValue=10000000.0;
248     for (int h=0;h<m;h++){
249         if (pois[h]<minValue){
250             minPlace=h;
251             minValue=pois[h];
252         }else if (pois[h] ==minValue){
253             Integer [] ran=new Integer [2];
254             ran[0]=h;
255             ran[1]=minPlace;
256             minPlace=getRandom(ran);
257         }
258     }
259     movement[minPlace] = movement[minPlace] +1.0;
260 }
261
262 return movement;
263 }
264
265 public static int getRandom(Integer [] array) {
266     int rnd = new Random().nextInt(array.length);
267     return array[rnd];
268 }
269
270 public static void writeMatrix(String filename, Double [][] yp, int m){
271
272     String fileName = filename;
273
274     try {
275         // New Filewriter
276         FileWriter fileWriter = new FileWriter(fileName);
277         BufferedWriter bufferedWriter =new BufferedWriter(fileWriter);
278
279         for (int i=0;i<m;i++){
280             for (int j=0; j<m; j++){
281                 bufferedWriter.write(String.valueOf(yp[i][j]));
282                 bufferedWriter.write(" ");
283             }
284             bufferedWriter.newLine();

```

```

285     }
286
287     // Close files
288     bufferedWriter.close();
289 }
290 catch(IOException ex) {
291     System.out.println(
292         "Error writing to file '"
293         + fileName + "'");
294 }
295 }
296
297 public static double round(double value, int places) {
298     if (places < 0) throw new IllegalArgumentException();
299
300     long factor = (long) Math.pow(10, places);
301     value = value * factor;
302     long tmp = Math.round(value);
303     return (double) tmp / factor;
304 }
305
306 }

```

7.5 CPLEX code for the Stochastic Network Flow Model

Listing 3: CPLEX code Stochastic Network Flow Model with input data Rio De Janeiro

```

1 import java.io.BufferedReader;
2 import java.io.BufferedWriter;
3 import java.io.FileNotFoundException;
4 import java.io.FileReader;
5 import java.io.FileWriter;
6 import java.io.IOException;
7
8 import ilog.concert.*;
9 import ilog.cplex.*;
10
11
12 public class ModelSNFRio {
13     public static void solveMe() throws FileNotFoundException, IOException{
14         //Initialization
15         double B=0.0;
16         int nd=1; //Denote number of days
17         int np=33; //Denote number of periods on a day
18         int NT=nd*np;
19         double NID=(double) NT;
20         int m=35; // Number of nodes in the system
21         Double[][][] r = new Double[m][m][NT]; //Create new 'matrix' for the arrival
           rates
22         //Because there are only four matrices, it is faster to read all matrices into
           the program and then getting values from the arrays[][]
23         Double[][] rm = DataSet("ODMetroRioMorningt.txt", m);
24         Double[][] rl = DataSet("ODMetroRioLuncht.txt", m);
25         Double[][] re = DataSet("ODMetroRioEveningt.txt", m);
26         Double[][] rw = DataSet("ODMetroRioWeekdayt.txt", m);
27
28         for (int i=0;i<m;i++){
29             for(int j=0;j<m;j++){
30                 for(int d=0;d<nd ; d++){
31                     for (int t= 33*d; t<(33*d)+6;t++){
32                         r[i][j][t]=(1.0/6.0)*rm[i][j]; // Fills the morning part on each day
33                     }
34                     for (int t= (33*d)+6; t<(33*d)+9;t++){
35                         r[i][j][t]=(3.0/28.0)*rw[i][j]; // Fills part between morning and
                           lunch on each day
36                     }
37                     for (int t= (33*d)+9; t<(33*d)+15;t++){
38                         r[i][j][t]=(1.0/6.0)*rl[i][j]; // Fills lunch part on each day
39                     }
40                     for (int t= (33*d)+15; t<(33*d)+21;t++){

```

```

41         r[i][j][t]=(6.0/28.0)*rw[i][j]; // Fills part between lunch and
           evening on each day
42     }
43     for (int t= (33*d)+21; t<(33*d)+27;t++){
44         r[i][j][t]=(1.0/6.0)*re[i][j]; // Fills evening part on each day
45     }
46     for (int t= (33*d)+27; t<(33*d)+33;t++){
47         r[i][j][t]=(6.0/28.0)*rw[i][j]; // Fills part after evening on each
           day
48     }
49 }
50 }
51 }
52
53
54 try{
55     //Define new model
56     IloCplex cplex = new IloCplex();
57
58     // Decision variables
59     IloNumVar [][] y = new IloNumVar[m][m][NT];
60     IloNumVar [] yi = new IloNumVar[m][NT];
61     for (int i=0; i<m; i++){
62         yi[i]=cplex.numVarArray(NT, 0.0, Double.MAX_VALUE);
63         for (int j=0; j<m; j++){
64             y[i][j]=cplex.numVarArray(NT, 0.0, Double.MAX_VALUE);
65         }
66     }
67     IloNumVar [] x = new IloNumVar[m];
68     x=cplex.numVarArray(m, 0.0, Double.MAX_VALUE);
69
70
71
72
73
74     //Expression for the summations and the objective
75     IloLinearNumExpr [][] typeDepartures = new IloLinearNumExpr[m][NT]; //
           Initialization for all the departures from station i, separately for all
           destinations
76     IloLinearNumExpr [][] typeArrivals = new IloLinearNumExpr[m][NT]; //
           Initialization for all the arrivals at station i, separately from all
           origins
77     IloLinearNumExpr [] C1r= new IloLinearNumExpr[m][NT]; // Initialization for
           the expression on the right side of constraint 1
78     IloLinearNumExpr [] C3r= new IloLinearNumExpr[m][NT]; // Initialization for
           the expression on the right side of constraint 3
79     IloLinearNumExpr objective=cplex.linearNumExpr(); // Initialization of the
           objective expression
80     for (int i=0; i<m; i++){
81         IloLinearNumExpr [] sumArrivals = new IloLinearNumExpr[NT];
82         IloLinearNumExpr [] sumDepartures = new IloLinearNumExpr[NT];
83         IloLinearNumExpr [] sumC1r = new IloLinearNumExpr[NT];
84         IloLinearNumExpr [] sumC3r = new IloLinearNumExpr[NT];
85         for (int t=0; t<NT; t++){
86             sumDepartures[t]=cplex.linearNumExpr();
87             sumArrivals[t]=cplex.linearNumExpr();
88             sumC1r[t]=cplex.linearNumExpr();
89             sumC3r[t]=cplex.linearNumExpr();
90             for (int j=0; j<m; j++){
91                 if (i!=j){
92                     sumDepartures[t].addTerm(1.0, y[i][j][t]);
93                     sumArrivals[t].addTerm(1.0, y[j][i][t]);
94                     sumC1r[t].addTerm(-1.0, y[i][j][t]);
95                     sumC1r[t].addTerm(1.0, y[j][i][t]);
96                     objective.addTerm(1.0, y[i][j][t]);
97                 }
98
99                 sumC3r[t].addTerm(1.0, y[i][j][t]);
100            }
101            sumC1r[t].addTerm(1.0, yi[i][t]);
102        }
103        typeArrivals[i]=sumArrivals;

```

```

104         typeDepartures [i]=sumDepartures;
105         C1r[i]=sumC1r;
106         C3r[i]=sumC3r;
107     }
108
109
110
111     // Define objective
112     cplex.addMaximize(objective);
113
114     //Constraint 1
115     for (int i=0;i<m;i++){
116         for (int t=0;t<NT-1;t++){
117             cplex.addEq(yi[i][t+1], C1r[i][t]);
118         }
119     }
120
121     //Constraint 2
122     cplex.addGe(objective , cplex.prod(B, cplex.sum(x)));
123
124
125
126     //Constraint 3
127     for(int i=0;i<m;i++){
128         for (int t=0;t<NT;t++){
129             cplex.addEq(yi[i][t], C3r[i][t]);
130         }
131     }
132
133     //Constraint 4
134     for(int i=0; i<m; i++){
135         for (int j=0;j<m;j++){
136             for (int l=0;l<m;l++){
137                 for(int t=0;t<NT;t++){
138                     if (i!=j && i!=l && j!=l)
139                         cplex.addEq(cplex.prod(y[i][j][t], r[i][l][t]), cplex.prod(y[i][l][t], r[i][j][t]));
140                 }
141             }
142         }
143     }
144
145     //Constraint 5
146     for(int i=0;i<m;i++){
147         cplex.addEq(yi[i][0], x[i]);
148     }
149
150     //Constraint 6
151     for(int i=0; i<m; i++){
152         for (int j=0;j<m;j++){
153             for (int t=0;t<NT;t++){
154                 if (j!=i){
155                     cplex.addLe(y[i][j][t], r[i][j][t]);
156                 }
157                 cplex.addGe(y[i][j][t], 0.0);
158             }
159         }
160     }
161
162
163     //Don't print what it wants to print
164     cplex.setParam(IloCplex.Param.Simplex.Display, 0);
165
166     // write model to file
167     cplex.exportModel("lpSNF.lp");
168
169
170
171     //Solve the problem and what to print if it can be solved
172     if(cplex.solve()){
173         Double [][] ya = new Double[m][m];
174         Double [] xp = new Double[m];

```

```

175         for (int i=0; i<m; i++){
176             xp[i]=cplex.getValue(x[i]);
177             for (int j =0; j<m; j++){
178                 Double[] temp = new Double[NT];
179                 for (int k=0; k<NT; k++){
180                     temp[k] = cplex.getValue(y[i][j][k]);
181                 }
182                 ya[i][j] = round(getAverage(temp), 2);
183             }
184         }
185
186         writeMatrix("SNFRio.txt", ya, m, xp);
187         System.out.println("objective = " + cplex.getObjValue());
188
189     }
190     else{
191         System.out.println("Problem not solved");
192     }
193     //End the problem
194     cplex.end();
195
196 }
197 catch (IloException e) {
198     System.err.println("Concert exception '" + e + "' caught");
199 }
200 }
201 }
202
203 public static double getAverage(Double[] x){
204     double average=0.0;
205     double sum=0.0;
206     for(int i=0; i<x.length; i++){
207         sum=sum+x[i];
208     }
209     average = sum/x.length;
210     return average;
211 }
212 public static Double [][] DataSet(String filename, int size) throws
213     FileNotFoundException, IOException {
214     String line = "";
215
216     Double [][] data = new Double[size][size];
217
218     FileReader fr= new FileReader(filename);
219     BufferedReader br = new BufferedReader(fr);
220     line=br.readLine();
221
222     String [] fline=line.split(";");
223     int c=fline.length;
224
225     for(int j =0; j<size; j++){
226         String [] theline=line.split(";");
227         Double [] td=new Double[size];
228         for (int i=0; i<c; i++){
229             td[i] = (double) ( Integer.parseInt(theline[i]));
230         }
231         data[j]=td;
232         if(j!=size){
233             line=br.readLine();
234         }
235     }
236 }
237
238 br.close();
239
240 return data;
241
242 }
243
244 }

```



```

246 public static void writeMatrix(String filename, Double [][] yp, int m, Double [] xp)
    {
247
248     String fileName = filename;
249
250     try {
251         // New FileWriter
252         FileWriter fileWriter = new FileWriter(fileName);
253         BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
254
255         for (int i=0; i<m; i++){
256             for (int j=0; j<m; j++){
257                 bufferedWriter.write(String.valueOf(yp[i][j]));
258                 bufferedWriter.write(" ");
259             }
260             bufferedWriter.newLine();
261         }
262         for (int i=0; i<m; i++){
263             bufferedWriter.write(String.valueOf(xp[i]));
264             bufferedWriter.write(" ");
265         }
266
267         // Close files
268         bufferedWriter.close();
269     }
270     catch(IOException ex) {
271         System.out.println(
272             "Error writing to file '"
273             + fileName + "'");
274     }
275 }
276
277 public static double round(double value, int places) {
278     if (places < 0) throw new IllegalArgumentException();
279
280     long factor = (long) Math.pow(10, places);
281     value = value * factor;
282     long tmp = Math.round(value);
283     return (double) tmp / factor;
284 }
285 }

```

7.6 CPLEX code for the Equilibrium State Model

Listing 4: CPLEX code Equilibrium State Model with input data Rio De Janeiro

```

1 import java.io.BufferedReader;
2 import java.io.BufferedWriter;
3 import java.io.FileNotFoundException;
4 import java.io.FileReader;
5 import java.io.FileWriter;
6 import java.io.IOException;
7 import java.text.DecimalFormat;
8
9 import ilog.concert.*;
10 import ilog.cplex.*;
11
12 public class ModelRio {
13     public static void solveMe() throws FileNotFoundException, IOException{
14         //Initialization
15         int N=600; // Number of bicycles in the system
16         int m=35; // Number of nodes in the system
17         Double [][] r = new Double[m][m]; //Create new 'matrix' for the arrival rates
18         r=DataSet("ODMetroRioEveningt.txt",35);
19         // Enter numerical example
20
21
22         try{
23
24             //Define new model

```

```

25     IloCplex cplex = new IloCplex();
26
27     // Decision variables
28     IloNumVar [][] y = new IloNumVar[m][m];
29     for (int i=0; i<m; i++){
30         y[i]=cplex.numVarArray(m, 0.0, Double.MAX_VALUE);
31     }
32
33     //Expression for the summations and the objective
34     IloLinearNumExpr [] typeDepartures = new IloLinearNumExpr[m]; //Initialization
35         for all the departures from station i, separately for all destinations
36     IloLinearNumExpr [] typeArrivals = new IloLinearNumExpr[m]; //Initialization
37         for all the arrivals at station i, separately from all origins
38     IloLinearNumExpr [] allBicycles = new IloLinearNumExpr[m]; //Initialization of
39         the numbers needed for constraint 4. For each i y-ii plus typeDepartures
40     IloLinearNumExpr objective=cplex.linearNumExpr();//Initialization of the
41         objective expression
42     for (int i=0; i<m; i++){
43         typeDepartures [i]=cplex.linearNumExpr();
44         typeArrivals [i]=cplex.linearNumExpr();
45         allBicycles [i]=cplex.linearNumExpr();
46         for (int j=0; j<m; j++){
47             if (i!=j){
48                 typeDepartures [i].addTerm(1.0, y[i][j]);
49                 typeArrivals [i].addTerm(1.0, y[j][i]);
50                 objective.addTerm(1.0, y[i][j]);
51             }
52             allBicycles [i].addTerm(1.0, y[i][j]);
53         }
54     }
55
56     // Define objective
57     cplex.addMaximize(objective);
58
59     //Constraint 1
60     for (int i=0; i<m; i++){
61         cplex.addEq(typeDepartures[i], typeArrivals[i]);
62     }
63
64     //Constraint 2
65     for (int i=0; i<m; i++){
66         for (int j=0; j<m; j++){
67             for (int l=0; l<m; l++){
68                 if (i!=j && i!=l ){
69                     cplex.addEq(cplex.prod(y[i][j], r[i][l]), cplex.prod(y[i][l], r[i][j]));
70                 }
71             }
72         }
73     }
74
75     //Constraint 3
76     for (int i=0; i<m; i++){
77         for (int j=0; j<m; j++){
78             if (i!=j){
79                 cplex.addLe(y[i][j], r[i][j]);
80             }
81             cplex.addGe(y[i][j], 0.0);
82         }
83     }
84
85     //Constraint 4
86     cplex.addEq(cplex.sum(allBicycles), N);
87
88     //Don't print what it wants to print
89     cplex.setParam(IloCplex.Param.Simplex.Display, 0);
90
91     // write model to file
92     cplex.exportModel("lpES.lp");

```

```

92 //Solve the problem and what to print if it can be solved
93 if(cplex.solve()){
94     Double [][] yp = new Double[m][m];
95     for(int i=0;i<m;i++){
96         for (int j =0; j<m; j++){
97             yp[i][j]=round(cplex.getValue(y[i][j]),2);
98         }
99     }
100     writeMatrix("ESRio.txt", yp, m);
101     System.out.println("Objective =" +cplex.getObjValue());
102     System.out.println("Bicycles station 1 =" + cplex.getValue(cplex.sum(y[0])
103         ));
104     System.out.println("Bicycles station 2 =" + cplex.getValue(cplex.sum(y[1])
105         ));
106     System.out.println("Bicycles station 3 =" + cplex.getValue(cplex.sum(y[2])
107         ));
108 }
109 else{
110     System.out.println("Problem not solved");
111 }
112 //End the problem
113 cplex.end();
114 }
115 catch (IOException e) {
116     System.err.println("Concert exception '" + e + "' caught");
117 }
118 }
119 //Read in the data
120
121
122
123
124 public static Double [][] DataSet(String filename, int size) throws
125     FileNotFoundException, IOException {
126     String line = "";
127
128     Double [][] data = new Double[size][size];
129
130     FileReader fr= new FileReader(filename);
131     BufferedReader br = new BufferedReader(fr);
132     line=br.readLine();
133
134     String [] fline=line.split(";");
135     int c=fline.length;
136
137
138     for(int j =0; j<size; j++){
139         String [] theline=line.split(";");
140         Double [] td=new Double[size];
141         for (int i=0; i<c; i++){
142             td[i] = (double) ( Integer.parseInt(theline[i]));
143         }
144         data[j]=td;
145         if(j!=size){
146             line=br.readLine();
147         }
148     }
149
150
151     br.close();
152
153     return data;
154
155
156 }
157
158 public static void writeMatrix(String filename, Double [][] yp, int m){
159

```

```

160     String fileName = filename;
161
162
163     try {
164         // New Filewriter
165         FileWriter fileWriter = new FileWriter(fileName);
166         BufferedWriter bufferedWriter =new BufferedWriter(fileWriter);
167
168         for (int i=0;i<m;i++){
169             for (int j=0; j<m; j++){
170                 bufferedWriter.write(String.valueOf(yp[i][j]));
171                 bufferedWriter.write(" ");
172             }
173             bufferedWriter.newLine();
174         }
175
176         // Close files
177         bufferedWriter.close();
178     }
179     catch(IOException ex) {
180         System.out.println(
181             "Error writing to file '"
182             + fileName + "'");
183     }
184 }
185
186 public static double round(double value, int places) {
187     if (places < 0) throw new IllegalArgumentException();
188
189     long factor = (long) Math.pow(10, places);
190     value = value * factor;
191     long tmp = Math.round(value);
192     return (double) tmp / factor;
193 }
194 }

```

7.7 Simulation with Anylogic

Listing 5: The Java source code for the Anylogic simulation

```

1
2 public class Anylogic {
3     package bbs_constaint_rates;
4
5     import java.io.Serializable;
6     import java.sql.Connection;
7     import java.sql.SQLException;
8     import java.util.ArrayDeque;
9     import java.util.ArrayList;
10    import java.util.Arrays;
11    import java.util.Calendar;
12    import java.util.Collection;
13    import java.util.Collections;
14    import java.util.Comparator;
15    import java.util.Currency;
16    import java.util.Date;
17    import java.util.Enumeration;
18    import java.util.HashMap;
19    import java.util.HashSet;
20    import java.util.Hashtable;
21    import java.util.Iterator;
22    import java.util.LinkedHashMap;
23    import java.util.LinkedHashSet;
24    import java.util.LinkedList;
25    import java.util.List;
26    import java.util.ListIterator;
27    import java.util.Locale;
28    import java.util.Map;
29    import java.util.PriorityQueue;
30    import java.util.Random;

```

```

31 import java.util.Set;
32 import java.util.SortedMap;
33 import java.util.SortedSet;
34 import java.util.Stack;
35 import java.util.Timer;
36 import java.util.TreeMap;
37 import java.util.TreeSet;
38 import java.util.Vector;
39 import java.awt.Color;
40 import java.awt.Font;
41 import java.awt.Graphics2D;
42 import java.awt.geom.AffineTransform;
43 import com.anylogic.engine.connectivity.ResultSet;
44 import com.anylogic.engine.connectivity.Statement;
45 import com.anylogic.engine.elements.IElementDescriptor;
46 import com.anylogic.engine.markup.Network;
47 import com.anylogic.engine.Position;
48 import com.anylogic.engine.markup.PedFlowStatistics;
49 import com.anylogic.engine.markup.DensityMap;
50
51 import static java.lang.Math.*;
52 import static com.anylogic.engine.UtilitiesArray.*;
53 import static com.anylogic.engine.UtilitiesCollection.*;
54 import static com.anylogic.engine.presentation.UtilitiesColor.*;
55 import static com.anylogic.engine.presentation.UtilitiesDrawing.*;
56 import static com.anylogic.engine.HyperArray.*;
57
58 import com.anylogic.engine.*;
59 import com.anylogic.engine.analysis.*;
60 import com.anylogic.engine.connectivity.*;
61 import com.anylogic.engine.gis.*;
62 import com.anylogic.engine.markup.*;
63 import com.anylogic.engine.presentation.*;
64
65 import com.anylogic.libraries.processmodeling.*;
66
67 import java.awt.geom.Arc2D;
68
69 import java.io.*;
70
71 public class Main extends Agent
72 {
73     // Parameters
74     // Plain Variables
75
76     public
77     double
78     NS1;
79     public
80     double
81     NS2;
82     public
83     double
84     NS3;
85     @AnyLogicInternalCodegenAPI
86     private static Map<String, IElementDescriptor> elementDescriptors_xjal = null;
87
88     @AnyLogicInternalCodegenAPI
89     @Override
90     public Map<String, IElementDescriptor> getElementDescriptors() {
91         if (elementDescriptors_xjal == null) {
92             elementDescriptors_xjal = createElementDescriptors(super.getElementDescriptors
93                 (), Main.class);
94         }
95         return elementDescriptors_xjal;
96     }
97     @AnyLogicCustomProposalPriority(type = AnyLogicCustomProposalPriority.Type.
98         STATIC_ELEMENT)
99     public static final Scale scale = new Scale( 10.0 );
100
101     @Override
102     public Scale getScale() {

```

```

101     return scale;
102 }
103 // Events
104
105 @AnyLogicInternalCodegenAPI
106 public EventTimeout _Station1DS_autoUpdateEvent_xjal = new EventTimeout(this);
107 @AnyLogicInternalCodegenAPI
108 public EventTimeout _Station2DS_autoUpdateEvent_xjal = new EventTimeout(this);
109 @AnyLogicInternalCodegenAPI
110 public EventTimeout _Station3DS_autoUpdateEvent_xjal = new EventTimeout(this);
111 @AnyLogicInternalCodegenAPI
112 public EventTimeout _statistics1_autoUpdateEvent_xjal = new EventTimeout(this);
113 @AnyLogicInternalCodegenAPI
114 public EventTimeout _statistics2_autoUpdateEvent_xjal = new EventTimeout(this);
115 @AnyLogicInternalCodegenAPI
116 public EventTimeout _statistics3_autoUpdateEvent_xjal = new EventTimeout(this);
117 @AnyLogicInternalCodegenAPI
118 public EventTimeout _chart_autoUpdateEvent_xjal = new EventTimeout(this);
119
120 @Override
121 @AnyLogicInternalCodegenAPI
122 public String getNameOf( EventTimeout _e ) {
123     if( _e == _Station1DS_autoUpdateEvent_xjal ) return "Station1DS auto update
124     event";
125     if( _e == _Station2DS_autoUpdateEvent_xjal ) return "Station2DS auto update
126     event";
127     if( _e == _Station3DS_autoUpdateEvent_xjal ) return "Station3DS auto update
128     event";
129     if( _e == _statistics1_autoUpdateEvent_xjal ) return "statistics1 auto update
130     event";
131     if( _e == _statistics2_autoUpdateEvent_xjal ) return "statistics2 auto update
132     event";
133     if( _e == _statistics3_autoUpdateEvent_xjal ) return "statistics3 auto update
134     event";
135     if( _e == _chart_autoUpdateEvent_xjal ) return "chart auto update event";
136     return super.getNameOf( _e );
137 }
138
139 @Override
140 @AnyLogicInternalCodegenAPI
141 public EventTimeout.Mode getModeOf( EventTimeout _e ) {
142     if ( _e == _Station1DS_autoUpdateEvent_xjal ) return EVENT.TIMEOUT_MODE.CYCLIC;
143     if ( _e == _Station2DS_autoUpdateEvent_xjal ) return EVENT.TIMEOUT_MODE.CYCLIC;
144     if ( _e == _Station3DS_autoUpdateEvent_xjal ) return EVENT.TIMEOUT_MODE.CYCLIC;
145     if ( _e == _statistics1_autoUpdateEvent_xjal ) return EVENT.TIMEOUT_MODE.CYCLIC
146     ;
147     if ( _e == _statistics2_autoUpdateEvent_xjal ) return EVENT.TIMEOUT_MODE.CYCLIC
148     ;
149     if ( _e == _statistics3_autoUpdateEvent_xjal ) return EVENT.TIMEOUT_MODE.CYCLIC
150     ;
151     if ( _e == _chart_autoUpdateEvent_xjal ) return EVENT.TIMEOUT_MODE.CYCLIC;
152     return super.getModeOf( _e );
153 }
154
155 @Override
156 @AnyLogicInternalCodegenAPI
157 public double getFirstOccurrenceTime( EventTimeout _e ) {
158     double _t;
159     if ( _e == _Station1DS_autoUpdateEvent_xjal ) {
160         _t =
161         0
162         ;
163         _t = toModelTime( _t , SECOND );
164         return _t;
165     }
166     if ( _e == _Station2DS_autoUpdateEvent_xjal ) {
167         _t =
168         0
169         ;
170         _t = toModelTime( _t , SECOND );
171         return _t;
172     }

```

```

164     if ( _e == _Station3DS_autoUpdateEvent_xjal ) {
165         _t =
166     0
167     ;
168         _t = toModelTime( _t , SECOND );
169         return _t;
170     }
171     if ( _e == _statistics1_autoUpdateEvent_xjal ) {
172         _t =
173     0
174     ;
175         _t = toModelTime( _t , SECOND );
176         return _t;
177     }
178     if ( _e == _statistics2_autoUpdateEvent_xjal ) {
179         _t =
180     0
181     ;
182         _t = toModelTime( _t , SECOND );
183         return _t;
184     }
185     if ( _e == _statistics3_autoUpdateEvent_xjal ) {
186         _t =
187     0
188     ;
189         _t = toModelTime( _t , SECOND );
190         return _t;
191     }
192     if ( _e == _chart_autoUpdateEvent_xjal ) {
193         _t =
194     0
195     ;
196         _t = toModelTime( _t , SECOND );
197         return _t;
198     }
199     return super.getFirstOccurrenceTime( _e );
200 }
201
202 @Override
203 @AnyLogicInternalCodegenAPI
204 public double evaluateTimeoutOf( EventTimeout _e ) {
205     double _t;
206     if( _e == _Station1DS_autoUpdateEvent_xjal ) {
207         _t =
208     1
209     ;
210         _t = toModelTime( _t , SECOND );
211         return _t;
212     }
213     if( _e == _Station2DS_autoUpdateEvent_xjal ) {
214         _t =
215     1
216     ;
217         _t = toModelTime( _t , SECOND );
218         return _t;
219     }
220     if( _e == _Station3DS_autoUpdateEvent_xjal ) {
221         _t =
222     1
223     ;
224         _t = toModelTime( _t , SECOND );
225         return _t;
226     }
227     if( _e == _statistics1_autoUpdateEvent_xjal ) {
228         _t =
229     1
230     ;
231         _t = toModelTime( _t , SECOND );
232         return _t;
233     }
234     if( _e == _statistics2_autoUpdateEvent_xjal ) {
235         _t =

```

```

236     1
237     ;
238         _t = toModelTime( _t , SECOND );
239         return _t;
240     }
241     if( _e == _statistics3_autoUpdateEvent_xjal ) {
242         _t =
243     1
244     ;
245         _t = toModelTime( _t , SECOND );
246         return _t;
247     }
248     if( _e == _chart_autoUpdateEvent_xjal ) {
249         _t =
250     1
251     ;
252         _t = toModelTime( _t , SECOND );
253         return _t;
254     }
255     return super.evaluateTimeoutOf( _e );
256 }
257
258 @Override
259 @AnyLogicInternalCodegenAPI
260 public void executeActionOf( EventTimeout _e ) {
261     if ( _e == _Station1DS_autoUpdateEvent_xjal ) {
262         Station1DS.update();
263         return;
264     }
265     if ( _e == _Station2DS_autoUpdateEvent_xjal ) {
266         Station2DS.update();
267         return;
268     }
269     if ( _e == _Station3DS_autoUpdateEvent_xjal ) {
270         Station3DS.update();
271         return;
272     }
273     if ( _e == _statistics1_autoUpdateEvent_xjal ) {
274         statistics1.update();
275         return;
276     }
277     if ( _e == _statistics2_autoUpdateEvent_xjal ) {
278         statistics2.update();
279         return;
280     }
281     if ( _e == _statistics3_autoUpdateEvent_xjal ) {
282         statistics3.update();
283         return;
284     }
285     if ( _e == _chart_autoUpdateEvent_xjal ) {
286         chart.updateData();
287         return;
288     }
289     super.executeActionOf( _e );
290 }
291
292 /** Internal constant, shouldn't be accessed by user */
293 @AnyLogicInternalCodegenAPI
294 protected static final short _STATECHART_ELEMENT_NEXT_ID_xjal = 0;
295 // Embedded Objects
296
297 @AnyLogicInternalCodegenAPI
298 private static final AgentAnimationSettings _bicycles_animationSettings_xjal =
299     new AgentAnimationSettings(1000L, 1000000000L);
300
301 public String getNameOf( Agent ao ) {
302     return super.getNameOf( ao );
303 }
304
305 public AgentAnimationSettings getAnimationSettingsOf( Agent ao ) {
306     return super.getAnimationSettingsOf( ao );
307 }

```



```

307
308 public class _bicycles_Population extends AgentArrayList<Bicycle> {
309     _bicycles_Population( Agent owner ) {
310         super( owner );
311     }
312
313     @AnyLogicInternalCodegenAPI
314     public void callSetupParameters( Bicycle agent, int index ) {
315         setupParameters_bicycles_xjal( agent, index );
316     }
317
318     @AnyLogicInternalCodegenAPI
319     public void callCreate( Bicycle agent, int index ) {
320         create_bicycles_xjal( agent, index );
321     }
322
323     @AnyLogicInternalCodegenAPI
324     public boolean isPresentationEnabled() {
325         return true;
326     }
327
328
329     public int NStation1() {
330         return _bicycles_NStation1_xjal();
331     }
332
333     public int NStation2() {
334         return _bicycles_NStation2_xjal();
335     }
336
337     public int NStation3() {
338         return _bicycles_NStation3_xjal();
339     }
340 }
341
342 @AnyLogicCustomProposalType(value = AnyLogicCustomProposalType.Label.POPULATION,
343     customText = "Bicycle")
344 public _bicycles_Population bicycles = new _bicycles_Population( this );
345
346 public String getNameOf( AgentList<?> aolist ) {
347     if( aolist == bicycles ) return "bicycles";
348     return super.getNameOf( aolist );
349 }
350
351 public AgentAnimationSettings getAnimationSettingsOf( AgentList<?> aolist ) {
352     if( aolist == bicycles ) return _bicycles_animationSettings_xjal;
353     return super.getAnimationSettingsOf( aolist );
354 }
355
356 /**
357  * This method creates and adds new embedded object in the replicated embedded
358  * object collection bicycles<br>
359  * @return newly created embedded object
360  */
361 public Bicycle add_bicycles() {
362     int index = bicycles.size();
363     Bicycle _result_xjal = instantiate_bicycles_xjal( index );
364     bicycles.callSetupParameters( _result_xjal, index );
365     bicycles.callCreate( _result_xjal, index );
366     _result_xjal.start();
367     return _result_xjal;
368 }
369
370 /**
371  * This method creates and adds new embedded object in the replicated embedded
372  * object collection bicycles<br>
373  * This method uses given parameter values to setup created embedded object<br>
374  * Index of this new embedded object instance can be obtained through calling <code>bicycles.size()</code> method <strong>before</strong> this method is
375  * called
376  * @param Rate12
377  * @param Rate13

```

```

374     * @param Rate21
375     * @param Rate23
376     * @param Rate31
377     * @param Rate32
378     * @param Start1
379     * @param Start2
380     * @param Start3
381     * @return newly created embedded object
382     */
383     public Bicycle add_bicycles( double Rate12, double Rate13, double Rate21, double
        Rate23, double Rate31, double Rate32, double Start1, double Start2, double
        Start3 ) {
384         int index = bicycles.size();
385         Bicycle _result_xjal = instantiate_bicycles_xjal( index );
386         // Setup parameters
387         _result_xjal.markParametersAreSet();
388         _result_xjal.Rate12 = Rate12;
389         _result_xjal.Rate13 = Rate13;
390         _result_xjal.Rate21 = Rate21;
391         _result_xjal.Rate23 = Rate23;
392         _result_xjal.Rate31 = Rate31;
393         _result_xjal.Rate32 = Rate32;
394         _result_xjal.Start1 = Start1;
395         _result_xjal.Start2 = Start2;
396         _result_xjal.Start3 = Start3;
397         // Finish embedded object creation
398         bicycles.callCreate( _result_xjal, index );
399         _result_xjal.start();
400         return _result_xjal;
401     }
402
403     /**
404     * This method removes the given embedded object from the replicated embedded
        object collection bicycles<br>
405     * The given object is destroyed, but not immediately in common case.
406     * @param object the active object – element of replicated embedded object
        bicycles – which should be removed
407     * @return <code>true</code> if object was removed successfully, <code>>false</
        code> if it doesn't belong to bicycles
408     */
409     public boolean remove_bicycles( Bicycle object ) {
410         if( ! bicycles.remove( object ) ) {
411             return false;
412         }
413         object.removeFromFlowchart();
414         object.setDestroyed();
415         return true;
416     }
417
418     /**
419     * Creates an embedded object instance and adds it to the end of replicated
        embedded object list<br>
420     * <i>This method should not be called by user</i>
421     */
422     protected Bicycle instantiate_bicycles_xjal( final int index ) {
423         Bicycle _result_xjal = new Bicycle( getEngine(), this, bicycles );
424
425         bicycles._add( _result_xjal );
426
427         return _result_xjal;
428     }
429
430     /**
431     * Setups parameters of an embedded object instance<br>
432     * This method should not be called by user
433     */
434     private void setupParameters_bicycles_xjal( final Bicycle self, final int index )
        {
435         self.Rate12 =
436         2.0
437         ;
438         self.Rate13 =

```

```

439     3.0
440     ;
441     self.Rate21 =
442     1.0
443     ;
444     self.Rate23 =
445     1.0
446     ;
447     self.Rate31 =
448     1.0
449     ;
450     self.Rate32 =
451     2.0
452     ;
453     self.Start1 = self._Start1_DefaultValue_xjal();
454     self.Start2 = self._Start2_DefaultValue_xjal();
455     self.Start3 = self._Start3_DefaultValue_xjal();
456     }
457
458     /**
459     * Setups an embedded object instance<br>
460     * This method should not be called by user
461     */
462     private void create_bicycles_xjal(Bicycle self, final int index ) {
463     self.setEnvironment( this );
464     self.setXYZ( 400.0, 60.0, 0.0 );
465     self.create();
466
467     // Port connections
468     }
469
470     /**
471     * <i>This method should not be called by user</i>
472     */
473     private int _bicycles_NStation1_xjal() {
474     int _value = 0;
475     for ( Bicycle item : bicycles ) {
476     boolean _t =
477     item.inState(Bicycle.Station1)
478     ;
479     if ( _t ) {
480     _value++;
481     }
482     }
483     return _value;
484     }
485     /**
486     * <i>This method should not be called by user</i>
487     */
488     private int _bicycles_NStation2_xjal() {
489     int _value = 0;
490     for ( Bicycle item : bicycles ) {
491     boolean _t =
492     item.inState(Bicycle.Station2)
493     ;
494     if ( _t ) {
495     _value++;
496     }
497     }
498     return _value;
499     }
500     /**
501     * <i>This method should not be called by user</i>
502     */
503     private int _bicycles_NStation3_xjal() {
504     int _value = 0;
505     for ( Bicycle item : bicycles ) {
506     boolean _t =
507     item.inState(Bicycle.Station3)
508     ;
509     if ( _t ) {
510     _value++;

```

```

511     }
512   }
513   return _value;
514 }
515 // Analysis Data Elements
516 @AnyLogicInternalCodegenAPI
517 public DataSet _chart_expression0_dataSet_xjal = new DataSet( 51, new
    DataUpdater_xjal() {
518     double _lastUpdateX = Double.NaN;
519     @Override
520     public void update( DataSet _d ) {
521         if ( time() == _lastUpdateX ) { return; }
522         _d.add( time(), _chart_expression0_dataSet_xjal_YValue() );
523         _lastUpdateX = time();
524     }
525 } );
526 /**
527  * <i>This method should not be called by user</i>
528  */
529 @AnyLogicInternalCodegenAPI
530 private double --chart_expression0_dataSet_xjal_YValue() {
531     return
532     bicycles.NStation1()
533 ;
534 }
535
536 @AnyLogicInternalCodegenAPI
537 public DataSet _chart_expression1_dataSet_xjal = new DataSet( 51, new
    DataUpdater_xjal() {
538     double _lastUpdateX = Double.NaN;
539     @Override
540     public void update( DataSet _d ) {
541         if ( time() == _lastUpdateX ) { return; }
542         _d.add( time(), _chart_expression1_dataSet_xjal_YValue() );
543         _lastUpdateX = time();
544     }
545 } );
546 /**
547  * <i>This method should not be called by user</i>
548  */
549 @AnyLogicInternalCodegenAPI
550 private double --chart_expression1_dataSet_xjal_YValue() {
551     return
552     bicycles.NStation2()
553 ;
554 }
555
556 @AnyLogicInternalCodegenAPI
557 public DataSet _chart_expression2_dataSet_xjal = new DataSet( 51, new
    DataUpdater_xjal() {
558     double _lastUpdateX = Double.NaN;
559     @Override
560     public void update( DataSet _d ) {
561         if ( time() == _lastUpdateX ) { return; }
562         _d.add( time(), _chart_expression2_dataSet_xjal_YValue() );
563         _lastUpdateX = time();
564     }
565 } );
566 /**
567  * <i>This method should not be called by user</i>
568  */
569 @AnyLogicInternalCodegenAPI
570 private double --chart_expression2_dataSet_xjal_YValue() {
571     return
572     bicycles.NStation3()
573 ;
574 }
575
576 public DataSet Station1DS = new DataSet( 50, new DataUpdater_xjal() {
577     double _lastUpdateX = Double.NaN;
578     @Override
579     public void update( DataSet _d ) {

```

```

580         if ( time() == _lastUpdateX ) { return; }
581         _d.add( time(), _Station1DS_YValue() );
582         _lastUpdateX = time();
583     }
584     @Override
585     public double getDataXValue() {
586         return time();
587     }
588 } );
589
590 /**
591  * <i>This method should not be called by user</i>
592  */
593 @AnyLogicInternalCodegenAPI
594 private double _Station1DS_YValue() {
595     return
596 bicycles.NStation1()
597 ;
598 }
599
600 public DataSet Station2DS = new DataSet( 50, new DataUpdater_xjal() {
601     double _lastUpdateX = Double.NaN;
602     @Override
603     public void update( DataSet _d ) {
604         if ( time() == _lastUpdateX ) { return; }
605         _d.add( time(), _Station2DS_YValue() );
606         _lastUpdateX = time();
607     }
608     @Override
609     public double getDataXValue() {
610         return time();
611     }
612 } );
613
614 /**
615  * <i>This method should not be called by user</i>
616  */
617 @AnyLogicInternalCodegenAPI
618 private double _Station2DS_YValue() {
619     return
620 bicycles.NStation2()
621 ;
622 }
623
624 public DataSet Station3DS = new DataSet( 50, new DataUpdater_xjal() {
625     double _lastUpdateX = Double.NaN;
626     @Override
627     public void update( DataSet _d ) {
628         if ( time() == _lastUpdateX ) { return; }
629         _d.add( time(), _Station3DS_YValue() );
630         _lastUpdateX = time();
631     }
632     @Override
633     public double getDataXValue() {
634         return time();
635     }
636 } );
637
638 /**
639  * <i>This method should not be called by user</i>
640  */
641 @AnyLogicInternalCodegenAPI
642 private double _Station3DS_YValue() {
643     return
644 bicycles.NStation3()
645 ;
646 }
647
648 public StatisticsContinuous statistics1 = new StatisticsContinuous( new
649     DataUpdater_xjal() {
650         double _lastUpdateX = Double.NaN;
651         @Override

```

```

651     public void update( StatisticsContinuous _d ) {
652         if ( time() == _lastUpdateX ) { return; }
653         _d.add( _statistics1_Value(), time() );
654         _lastUpdateX = time();
655     }
656 } );
657
658 /**
659  * <i>This method should not be called by user</i>
660  */
661 @AnyLogicInternalCodegenAPI
662 private double _statistics1_Value() {
663     return
664 NS1
665 ;
666 }
667
668 public StatisticsContinuous statistics2 = new StatisticsContinuous( new
669     DataUpdater_xjal() {
670         double _lastUpdateX = Double.NaN;
671         @Override
672         public void update( StatisticsContinuous _d ) {
673             if ( time() == _lastUpdateX ) { return; }
674             _d.add( _statistics2_Value(), time() );
675             _lastUpdateX = time();
676         }
677     } );
678
679 /**
680  * <i>This method should not be called by user</i>
681  */
682 @AnyLogicInternalCodegenAPI
683 private double _statistics2_Value() {
684     return
685 NS2
686 ;
687 }
688
689 public StatisticsContinuous statistics3 = new StatisticsContinuous( new
690     DataUpdater_xjal() {
691         double _lastUpdateX = Double.NaN;
692         @Override
693         public void update( StatisticsContinuous _d ) {
694             if ( time() == _lastUpdateX ) { return; }
695             _d.add( _statistics3_Value(), time() );
696             _lastUpdateX = time();
697         }
698     } );
699
700 /**
701  * <i>This method should not be called by user</i>
702  */
703 @AnyLogicInternalCodegenAPI
704 private double _statistics3_Value() {
705     return
706 NS3
707 ;
708 }
709
710 // View areas
711 public ViewArea _origin_VA = new ViewArea( this, "[Origin]", 0, 0, ViewArea.
712     TOP_LEFT, ViewArea.SPECIFIED_ZOOM, 1, 100, 100 );
713
714 @Override
715 @AnyLogicInternalCodegenAPI
716 public int getViewAreas( Map<String, ViewArea> _output ) {
717     if ( _output != null ) {
718         _output.put( "_origin_VA", this._origin_VA );
719     }
720     return 1 + super.getViewAreas( _output );
721 }
722 @AnyLogicInternalCodegenAPI
723 protected static final int _bicycles_presentation = 1;

```

```

720     @AnyLogicInternalCodegenAPI
721     protected static final int _chart = 2;
722
723     /** Internal constant, shouldn't be accessed by user */
724     @AnyLogicInternalCodegenAPI
725     protected static final int _SHAPE_NEXT_ID_xjal = 3;
726
727
728     /**
729      * Top-level presentation group id
730      */
731     @AnyLogicInternalCodegenAPI
732     protected static final int _presentation = 0;
733
734     @AnyLogicInternalCodegenAPI
735     public boolean isPublicPresentationDefined() {
736         return true;
737     }
738
739     @AnyLogicInternalCodegenAPI
740     public boolean isEmbeddedAgentPresentationVisible( Agent _a ) {
741         return super.isEmbeddedAgentPresentationVisible( _a );
742     }
743     /**
744      * Top-level icon group id
745      */
746     @AnyLogicInternalCodegenAPI
747     protected static final int _icon = -1;
748
749
750     protected TimeStackChart chart;
751
752     /**
753      * <i>This method should not be called by user</i>
754      */
755     @AnyLogicInternalCodegenAPI
756     private void _bicycles_presentation_SetDynamicParams_xjal(
757         ShapeEmbeddedObjectPresentation shape, int index ) {
758         shape.setEmbeddedObject_xjal(
759             bicycles.get( index )
760         );
761     }
762
763     /**
764      * <i>This method should not be called by user</i>
765      */
766     @AnyLogicInternalCodegenAPI
767     protected ShapeEmbeddedObjectPresentation
768         _bicycles_presentation_createShapeWithStaticProperties_xjal( final int _index
769         ) {
770         ShapeEmbeddedObjectPresentation shape = new ShapeEmbeddedObjectPresentation(
771             Main.this, SHAPE_DRAW_2D3D, true, 400.0, 60.0, 0.0, 0.0,
772             true, true, bicycles.get( _index ) );
773         return shape;
774     }
775
776     /**
777      * <i>This method should not be called by user</i>
778      */
779     @AnyLogicInternalCodegenAPI
780     private int _bicycles_presentation_Replication() {
781         return
782         bicycles.size()
783     };
784
785     protected ReplicatedShape<ShapeEmbeddedObjectPresentation> bicycles_presentation;
786
787     private INetwork[] _getNetworks_xjal;
788
789     @Override

```

```

788     public INetwork[] getNetworks() {
789         return _getNetworks_xjal;
790     }
791
792
793     private com.anylogic.engine.markup.Ground[] _getGrounds_xjal;
794
795     @Override
796     public com.anylogic.engine.markup.Ground[] getGrounds() {
797         return _getGrounds_xjal;
798     }
799
800
801     private com.anylogic.engine.markup.RailwayNetwork[] _getRailwayNetworks_xjal;
802
803     @Override
804     public com.anylogic.engine.markup.RailwayNetwork[] getRailwayNetworks() {
805         return _getRailwayNetworks_xjal;
806     }
807
808     @AnyLogicInternalCodegenAPI
809     private void _createPersistentElementsBP0_xjal() {
810     }
811
812     @AnyLogicInternalCodegenAPI
813     private void _createPersistentElementsAP0_xjal() {
814         {
815             DataSet _item;
816             List<DataSet> _items = new ArrayList<DataSet>( 3 );
817             _items.add( _chart_expression0_dataSet_xjal );
818             _items.add( _chart_expression1_dataSet_xjal );
819             _items.add( _chart_expression2_dataSet_xjal );
820             List<String> _titles = new ArrayList<String>( 3 );
821             _titles.add( "Station1" );
822             _titles.add( "Station2" );
823             _titles.add( "Station3" );
824             List<Color> _colors = new ArrayList<Color>( 3 );
825             _colors.add( lavender );
826             _colors.add( gold );
827             _colors.add( yellowGreen );
828             chart = new TimeStackChart(
829                 Main.this, true, 40.0, 120.0,
830                 260.0, 210.0,
831                 null, null,
832                 50.0, 30.0,
833                 180.0, 120.0, white, black, black,
834                 30.0, Chart.SOUTH,
835
836     100
837                 , Chart.WINDOW_MOVES_WITH_TIME, null, Chart.SCALE_AUTO
838                 , 0, Chart.GRID_DEFAULT, Chart.GRID_DEFAULT,
839                 darkGray, darkGray, _items, _titles, _colors );
840         }
841         bicycles_presentation = new ReplicatedShape<ShapeEmbeddedObjectPresentation>()
842         {
843             @Override
844             public Class<ShapeEmbeddedObjectPresentation> getShapeClass() {
845                 return ShapeEmbeddedObjectPresentation.class;
846             }
847
848             @Override
849             public int getReplication() {
850                 return _bicycles_presentation_Replication();
851             }
852
853             @Override
854             public ShapeEmbeddedObjectPresentation createShapeWithStaticProperties_xjal(
855                 int index ) {
856                 ShapeEmbeddedObjectPresentation _e =
857                     _bicycles_presentation_createShapeWithStaticProperties_xjal( index );
858                 return _e;
859             }
860         }

```



```

857
858     @Override
859     public void setShapeDynamicProperties_xjal( ShapeEmbeddedObjectPresentation
            shape, int index ) {
860         _bicycles_presentation_SetDynamicParams_xjal( shape, index );
861     }
862 };
863 }
864
865
866 // Static initialization of persistent elements
867 {
868     _createPersistentElementsBP0_xjal();
869 }
870 protected ShapeTopLevelPresentationGroup presentation;
871 protected ShapeGroup icon;
872
873 @Override
874 @AnyLogicInternalCodegenAPI
875 public Object getPersistentShape( int _shape ) {
876     switch ( _shape ) {
877         case _presentation: return presentation;
878         case _icon: return icon;
879         case _chart: return chart;
880         case _bicycles_presentation: return bicycles_presentation;
881         default: return super.getPersistentShape( _shape );
882     }
883 }
884
885 @Override
886 @AnyLogicInternalCodegenAPI
887 public String getNameOfShape_xjal( Object _shape ) {
888     try {
889         if ( _shape == null ) return null;
890         String _name_xjal;
891         _name_xjal = checkNameOfShape_xjal( _shape, presentation, "presentation" );
892         if ( _name_xjal != null ) return _name_xjal;
893         _name_xjal = checkNameOfShape_xjal( _shape, icon, "icon" ); if ( _name_xjal !=
            null ) return _name_xjal;
894         _name_xjal = checkNameOfShape_xjal( _shape, chart, "chart" ); if ( _name_xjal
            != null ) return _name_xjal;
895         _name_xjal = checkNameOfShape_xjal( _shape, bicycles_presentation, "
            bicycles_presentation" ); if ( _name_xjal != null ) return _name_xjal;
896     } catch (Exception e) {
897         return null;
898     }
899     return super.getNameOfShape_xjal( _shape );
900 }
901
902 @AnyLogicInternalCodegenAPI
903 private void drawModelElements_PlainVariables_xjal( Panel _panel, Graphics2D _g,
            boolean _publicOnly, boolean _isSuperClass ) {
904     if ( !_publicOnly ) {
905         drawPlainVariable( _panel, _g, 240, 400, 10, 0, "NS1", NS1, false );
906     }
907     if ( !_publicOnly ) {
908         drawPlainVariable( _panel, _g, 240, 450, 10, 0, "NS2", NS2, false );
909     }
910     if ( !_publicOnly ) {
911         drawPlainVariable( _panel, _g, 240, 490, 10, 0, "NS3", NS3, false );
912     }
913 }
914
915 @AnyLogicInternalCodegenAPI
916 private void drawModelElements_DataElements_xjal( Panel _panel, Graphics2D _g,
            boolean _publicOnly, boolean _isSuperClass ) {
917     if ( !_publicOnly ) {
918         drawDataset( _panel, _g, 60, 400, 15, 0, "Station1DS", Station1DS );
919     }
920     if ( !_publicOnly ) {
921         drawDataset( _panel, _g, 60, 450, 15, 0, "Station2DS", Station2DS );
922     }

```

```

922     if (!_publicOnly) {
923         drawDataset( _panel, _g, 60, 500, 15, 0, "Station3DS", Station3DS );
924     }
925     if (!_publicOnly) {
926         drawStatistics( _panel, _g, 330, 400, 15, 0, "statistics1", statistics1 );
927     }
928     if (!_publicOnly) {
929         drawStatistics( _panel, _g, 330, 450, 15, 0, "statistics2", statistics2 );
930     }
931     if (!_publicOnly) {
932         drawStatistics( _panel, _g, 330, 500, 15, 0, "statistics3", statistics3 );
933     }
934 }
935
936 @AnyLogicInternalCodegenAPI
937 private void drawModelElements_EmbeddeObjects_xjal(Panel _panel, Graphics2D _g,
    boolean _publicOnly, boolean _isSuperClass ) {
938     // Embedded object "bicycles"
939     if (!_publicOnly) {
940         drawEmbeddedObjectModelDefault( _panel, _g, 100, 60, 10, 0, "bicycles",
            this.bicycles );
941     }
942 }
943
944 @AnyLogicInternalCodegenAPI
945 private void drawModelElements_AgentLinks_xjal(Panel _panel, Graphics2D _g,
    boolean _publicOnly, boolean _isSuperClass ) {
946     if (_publicOnly) { return; }
947     drawLinkToAgent( _panel, _g, 50, -50, 15, 0, "connections", true, connections
        );
948 }
949
950 @Override
951 @AnyLogicInternalCodegenAPI
952 public void drawModelElements( Panel _panel, Graphics2D _g, boolean _publicOnly,
    boolean _isSuperClass ) {
953     super.drawModelElements( _panel, _g, _publicOnly, true );
954     drawModelElements_PlainVariables_xjal( _panel, _g, _publicOnly, _isSuperClass )
        ;
955     drawModelElements_DataElements_xjal( _panel, _g, _publicOnly, _isSuperClass );
956     drawModelElements_EmbeddeObjects_xjal( _panel, _g, _publicOnly, _isSuperClass )
        ;
957     drawModelElements_AgentLinks_xjal( _panel, _g, _publicOnly, _isSuperClass );
958 }
959
960 @AnyLogicInternalCodegenAPI
961 private boolean onClickModelAt_EmbeddedObjects_xjal( Panel _panel, double _x,
    double _y, int _clickCount, boolean _publicOnly, boolean _isSuperClass ) {
962     if ( !bicycles.isEmpty() && modelElementContains(_x, _y, 100, 60) ) {
963         if ( _clickCount == 2 ) {
964             _panel.browseAgent_xjal( 100, 60, this, "bicycles" );
965         } else {
966             _panel.addInspect( 100, 60, this, "bicycles" );
967         }
968         return true;
969     }
970     return false;
971 }
972
973 @AnyLogicInternalCodegenAPI
974 private boolean onClickModelAt_AgentLinks_xjal( Panel _panel, double _x, double
    _y, int _clickCount, boolean _publicOnly, boolean _isSuperClass ) {
975     if ( modelElementContains(_x, _y, 50, -50) ) {
976         _panel.addInspect_xjal( 50, -50, this, "connections", Panel.
            INSPECT_CONNECTIONS_xjal );
977         return true;
978     }
979     return false;
980 }
981
982
983 @AnyLogicInternalCodegenAPI

```

```

984 private boolean onClickModelAt_PlainVariables_xjal( Panel _panel, double _x,
985     double _y, int _clickCount, boolean _publicOnly, boolean _isSuperClass ) {
986     if( !_publicOnly && modelElementContains(_x, _y, 240, 400) ) {
987         _panel.addInspect( 240, 400, this, "NS1" );
988         return true;
989     }
990     if( !_publicOnly && modelElementContains(_x, _y, 240, 450) ) {
991         _panel.addInspect( 240, 450, this, "NS2" );
992         return true;
993     }
994     if( !_publicOnly && modelElementContains(_x, _y, 240, 490) ) {
995         _panel.addInspect( 240, 490, this, "NS3" );
996         return true;
997     }
998     return false;
999 }
1000
1001 @AnyLogicInternalCodegenAPI
1002 private boolean onClickModelAt_DataElements_xjal( Panel _panel, double _x, double
1003     _y, int _clickCount, boolean _publicOnly, boolean _isSuperClass ) {
1004     if( !_publicOnly && modelElementContains(_x, _y, 60, 400) ) {
1005         _panel.addInspect( 60, 400, this, "Station1DS" );
1006         return true;
1007     }
1008     if( !_publicOnly && modelElementContains(_x, _y, 60, 450) ) {
1009         _panel.addInspect( 60, 450, this, "Station2DS" );
1010         return true;
1011     }
1012     if( !_publicOnly && modelElementContains(_x, _y, 60, 500) ) {
1013         _panel.addInspect( 60, 500, this, "Station3DS" );
1014         return true;
1015     }
1016     if( !_publicOnly && modelElementContains(_x, _y, 330, 400) ) {
1017         _panel.addInspect( 330, 400, this, "statistics1" );
1018         return true;
1019     }
1020     if( !_publicOnly && modelElementContains(_x, _y, 330, 450) ) {
1021         _panel.addInspect( 330, 450, this, "statistics2" );
1022         return true;
1023     }
1024     if( !_publicOnly && modelElementContains(_x, _y, 330, 500) ) {
1025         _panel.addInspect( 330, 500, this, "statistics3" );
1026         return true;
1027     }
1028     return false;
1029 }
1030
1031 @Override
1032 @AnyLogicInternalCodegenAPI
1033 public boolean onClickModelAt( Panel _panel, double _x, double _y, int
1034     _clickCount, boolean _publicOnly, boolean _isSuperClass ) {
1035     if ( onClickModelAt_EmbeddedObjects_xjal( _panel, _x, _y, _clickCount,
1036         _publicOnly, _isSuperClass ) ) { return true; }
1037     if ( onClickModelAt_AgentLinks_xjal( _panel, _x, _y, _clickCount, _publicOnly,
1038         _isSuperClass ) ) { return true; }
1039     if ( onClickModelAt_PlainVariables_xjal( _panel, _x, _y, _clickCount,
1040         _publicOnly, _isSuperClass ) ) { return true; }
1041     if ( onClickModelAt_DataElements_xjal( _panel, _x, _y, _clickCount, _publicOnly,
1042         _isSuperClass ) ) { return true; }
1043     return super.onClickModelAt( _panel, _x, _y, _clickCount, _publicOnly, true );
1044 }
1045
1046 /**
1047  * Constructor
1048  */
1049 public Main( Engine engine, Agent owner, AgentList<? extends Main>
1050     ownerPopulation ) {
1051     super( engine, owner, ownerPopulation );
1052     if ( isTopLevelClass_xjal( Main.class ) ) {
1053         instantiateBaseStructure_xjal();
1054     }
1055 }

```

```

1048     }
1049 }
1050
1051 @AnyLogicInternalCodegenAPI
1052 public void onOwnerChanged_xjal() {
1053     super.onOwnerChanged_xjal();
1054     setupReferences_xjal();
1055 }
1056
1057 @AnyLogicInternalCodegenAPI
1058 public void instantiateBaseStructure_xjal() {
1059     super.instantiateBaseStructure_xjal();
1060     setupReferences_xjal();
1061 }
1062
1063 @AnyLogicInternalCodegenAPI
1064 private void setupReferences_xjal() {
1065 }
1066
1067 /**
1068  * Simple constructor. Please add created agent to some population by calling
1069  * goToPopulation() function
1070  */
1071 public Main() {
1072 }
1073
1074 /**
1075  * Creating embedded object instances
1076  */
1077 @AnyLogicInternalCodegenAPI
1078 private void instantiatePopulations_xjal() {
1079     {
1080         int _cnt =
1081         10
1082         ;
1083         for ( int i = bicycles.size(); i < _cnt; i++ ) {
1084             instantiate_bicycles_xjal( i );
1085         }
1086     }
1087
1088 @Override
1089 @AnyLogicInternalCodegenAPI
1090 public void create() {
1091     super.create();
1092     // Creating embedded object instances
1093     instantiatePopulations_xjal();
1094     // Assigning initial values for plain variables
1095     setupPlainVariables_Main_xjal();
1096     // Dynamic initialization of persistent elements
1097     _createPersistentElementsAP0_xjal();
1098     presentation = new ShapeTopLevelPresentationGroup( Main.this, true, 0, 0, 0, 0
1099         , bicycles.presentation, chart );
1100     icon = new ShapeGroup( Main.this, true, 0, 0, 0 );
1101     // Creating contents for replicated shapes
1102     bicycles.presentation.createShapes();
1103     // Creating embedded object instances
1104     instantiatePopulations_xjal();
1105     // Environments setup
1106     {
1107         double _x_xjal =
1108         500
1109         ;
1110         double _y_xjal =
1111         500
1112         ;
1113         double _z_xjal =
1114         0
1115         ;
1116         setupSpace( _x_xjal, _y_xjal, _z_xjal );
1117     }
1118     disableSteps();

```

```

1118     setNetworkUserDefined();
1119     setLayoutType( LAYOUTRANDOM );
1120     // Port connectors with non-replicated objects
1121     // Creating replicated embedded objects
1122     bicycles.setEnvironment( this );
1123     for ( int i = 0; i < bicycles.size(); i++ ) {
1124         setupParameters_bicycles_xjal( bicycles.get(i), i );
1125         create_bicycles_xjal( bicycles.get(i), i );
1126     }
1127     setupInitialConditions_xjal( Main.class );
1128     if (isTopLevelClass_xjal( Main.class )) {
1129         onCreate();
1130     }
1131 }
1132
1133 @AnyLogicInternalCodegenAPI
1134 public void setupExt_xjal( AgentExtension _ext ) {
1135     // Agent properties setup
1136     if ( _ext instanceof ExtAgentWithSpatialMetrics && _ext instanceof
1137         ExtWithSpaceType ) {
1138         double _value;
1139         _value =
1140         10
1141         ;
1142         ((ExtAgentWithSpatialMetrics) _ext).setSpeed( _value, MPS );
1143     }
1144 }
1145
1146 @Override
1147 @AnyLogicInternalCodegenAPI
1148 public void start() {
1149     super.start();
1150     _Station1DS_autoUpdateEvent_xjal.start();
1151     _Station2DS_autoUpdateEvent_xjal.start();
1152     _Station3DS_autoUpdateEvent_xjal.start();
1153     _statistics1_autoUpdateEvent_xjal.start();
1154     _statistics2_autoUpdateEvent_xjal.start();
1155     _statistics3_autoUpdateEvent_xjal.start();
1156     _chart_autoUpdateEvent_xjal.start();
1157     applyLayout();
1158     for ( Agent embeddedObject : bicycles ){
1159         embeddedObject.start();
1160     }
1161     if (isTopLevelClass_xjal( Main.class )) {
1162         onStartUp();
1163     }
1164 }
1165
1166 /**
1167  * Assigning initial values for plain variables<br>
1168  * <em>This method isn't designed to be called by user and may be removed in
1169  * future releases.</em>
1170  */
1171 @AnyLogicInternalCodegenAPI
1172 public void setupPlainVariables_xjal() {
1173     setupPlainVariables_Main_xjal();
1174 }
1175
1176 /**
1177  * Assigning initial values for plain variables<br>
1178  * <em>This method isn't designed to be called by user and may be removed in
1179  * future releases.</em>
1180  */
1181 @AnyLogicInternalCodegenAPI
1182 private void setupPlainVariables_Main_xjal() {
1183     NS1 =
1184     0.0
1185     ;
1186     NS2 =
1187     0.0
1188     ;

```

```

1187     NS3 =
1188     0.0
1189     ;
1190     }
1191
1192     // User API -----
1193     @AnyLogicInternalCodegenAPI
1194     static LinkToAgentAnimationSettings _connections_commonAnimationSettings_xjal =
1195         new LinkToAgentAnimationSettingsImpl( false , black , 1.0 , LINE_STYLE_SOLID ,
1196             ARROW_NONE , 0.0 );
1197
1198     public LinkToAgentCollection<Agent , Agent> connections = new
1199         LinkToAgentStandardImpl<Agent , Agent>(this ,
1200             _connections_commonAnimationSettings_xjal);
1201
1202     @Override
1203     public LinkToAgentCollection<? extends Agent , ? extends Agent>
1204         getLinkToAgentStandard_xjal() {
1205         return connections;
1206     }
1207
1208     @AnyLogicInternalCodegenAPI
1209     public void drawLinksToAgents(boolean _underAgents_xjal , LinkToAgentAnimator
1210         _animator_xjal) {
1211         super.drawLinksToAgents(_underAgents_xjal , _animator_xjal);
1212         if ( _underAgents_xjal ) {
1213             _animator_xjal.drawLink( this , connections , true , true );
1214         }
1215     }
1216
1217     public List<Object> getEmbeddedObjects() {
1218         List<Object> list = super.getEmbeddedObjects();
1219         if (list == null) {
1220             list = new LinkedList<Object>();
1221         }
1222         list.add( bicycles );
1223         return list;
1224     }
1225
1226     public AgentList<? extends Main> getPopulation() {
1227         return (AgentList<? extends Main>) super.getPopulation();
1228     }
1229
1230     public List<? extends Main> agentsInRange( double distance ) {
1231         return (List<? extends Main>) super.agentsInRange( distance );
1232     }
1233
1234     @AnyLogicInternalCodegenAPI
1235     public void onDestroy() {
1236         _Station1DS_autoUpdateEvent_xjal.onDestroy();
1237         _Station2DS_autoUpdateEvent_xjal.onDestroy();
1238         _Station3DS_autoUpdateEvent_xjal.onDestroy();
1239         _statistics1_autoUpdateEvent_xjal.onDestroy();
1240         _statistics2_autoUpdateEvent_xjal.onDestroy();
1241         _statistics3_autoUpdateEvent_xjal.onDestroy();
1242         _chart_autoUpdateEvent_xjal.onDestroy();
1243         for (Agent _item : bicycles) {
1244             _item.onDestroy();
1245         }
1246     }
1247
1248     // Analysis Data Elements
1249     _chart_expression0_dataSet_xjal.destroyUpdater_xjal();
1250     _chart_expression1_dataSet_xjal.destroyUpdater_xjal();
1251     _chart_expression2_dataSet_xjal.destroyUpdater_xjal();
1252     Station1DS.destroyUpdater_xjal();
1253     Station2DS.destroyUpdater_xjal();
1254     Station3DS.destroyUpdater_xjal();
1255     statistics1.destroyUpdater_xjal();
1256     statistics2.destroyUpdater_xjal();
1257     statistics3.destroyUpdater_xjal();
1258     super.onDestroy();
1259 }

```

1253

1254

1255 }

1256

1257 }