UNIVERSITAT JAUME I

Development of a distributed system for real time traffic analysis using agents

by

Pablo Jiménez Mateo

A thesis submitted in partial fulfillment for the
degree of Master in Intelligent Systems

in the
Escola Superior de Tecnologia i Ciències Experimentals
Department of computer engineering

November 10, 2016

Declaration of Authorship

I, Pablo Jiménez Mateo, declare that this thesis titled, 'Development of a distributed system for real time traffic analysis using agents' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"Write a funny quote here."

If the quote is taken from someone, their name goes here

UNIVERSITAT JAUME I

Abstract

Escola Superior de Tecnologia i Ciències Experimentals
Department of computer engineering

Master in Intelligent Systems

by Pablo Jiménez Mateo

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

Contents

Declaration of Authorship i

Abstract iii

Acknowledgements iv

List of Figures vii

List of Tables viii

Abbreviations ix

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Objectives . . . 2
- 1.3 Planification . . . 3
- 1.4 Employed technologies . . . 4
- 1.5 Structure . . . 4

2 State of the art 5
- 2.1 Smart cities . . . 5
- 2.2 Smart transport . . . 5

3 Parts of the simulator 7
- 3.1 Simulator files overview . . . 7
  - 3.1.1 Agents . . . 7
  - 3.1.2 Behaviors . . . 8
  - 3.1.3 Environmental classes . . . 10
  - 3.1.4 CanvasWorld . . . 11
  - 3.1.5 Main . . . 11
  - 3.1.6 Search algorithms . . . 12
  - 3.1.7 Static files . . . 12
- 3.2 Communication ontology . . . 14

4 Routing algorithms 16

4.1 What is a routing algorithm 16
4.1.1 Dijkstra's algorithm 17
4.1.2 Bellman-Ford algorithm 17
4.1.3 A* algorithm 17
4.2 Chosen algorithms 17
4.2.1 Shortest path algorithm 17
4.2.2 Fastest path algorithm 18
4.2.3 Smart path algorithm 18

Bibliography **19**

List of Figures

3.1 Ontology communications scheme 14

List of Tables

1.1 Time planification 3

3.1 A few rows of the events.csv file 13
3.2 The complete list of ontologies 15

Abbreviations

API	**A**pplication **P**rogramming **I**nterface
GPS	**G**lobal **P**osition **S**ystem
GUI	**G**raphical **U**ser **I**nterface
IoT	**I**nternet **O**f **T**hings
WORA	**W**rite **O**nce **R**un **A**nywhere

For/Dedicated to/To my...

Chapter 1

Introduction

In this first chapter, the motivation of the project and its main objectives are stated. After that the time planification, employed technologies and the structure of the rest of the thesis are presented.

1.1 Motivation

Nowadays, with the penetration of new technologies such as smartphones and self driving cars, it is possible to share information about the state of the traffic on the transportation networks. This is very useful to have real time path finding algorithms that avoid congestion.

Furthermore, vehicle to vehicle and vehicle to infrastructure communications has been researched for a long time[1] and is still being researched nowadays [2], and this is a direct application of it.

One of the main concerns with this technology is its possible use for surveillance purposes, given the kind of data this system collects and uses privacy should not be overlooked.

The motivation of this project comes from the necessity of developing a tool that allows us to study the behavior of simulated agents in a real scenario. This system is distributed due to its nature, and keeps the privacy of the users at every moment.

The motivation of this project comes from the necessity of studying autonomous vehicles and how they behave depending on traffic. To achieve this goal a multiagent system that supports the usage of distributed agents will be developed. On it, multiple rerouting algorithms will be tested based on [3] and results will be analyzed based on the real time state of the traffic.

1.2 Objectives

The main goals of the final master project are:

- Development of a simulator to test traffic routing algorithms
- Analyze the existing algorithms for traffic routing
- Propose an algorithm that analyzes the traffic in real time
- Study and analysis of the proposed algorithm

1.3 Planification

Task	**Planned hours**	**Goal**
Study and installation of the JADE library	10 hours	Understand how the library JADE works
Development of a graph to represent the urban network	5 hours	Complete the classes and files needed to represent the network
Development of a communication protocol and its ontologies	20 hours	Understand how the inter agent communication will be done
Development of agents	-	-
- Development of the first movile agents with JADE	15 hours	Implement the vehicles
- Development of the behaviur of the vehicles	15 hours	Implement the logic of the vehicles
- Development of segment agents	15 hours	Implement the segments
- Development of the agent that keeps the time	10 hours	Implement a reliable and deterministic model of time
- Development of the agent that launches the events	10 hours	Implement an agent able to read events from file and launch them to the simulator
Development of the graphical user interface	25 hours	Implement a graphical user interface to view the results in real time
Development of routing algorithms	35 hours	Implement various routing algorithms
Testing and optimizing the application	40 hours	Stress testing the application
Gathering of results	10 hours	Running simulations and creating the graphs
Documentation	10 hours	Documentation of the communication protocols, ontologies and classes
Writing the thesis	80 hours	Writing of the thesis
TOTAL	**300 hours**	

TABLE 1.1: Time planification

1.4 Employed technologies

The language of choice for this project was Java [4], Java is an all purpose programming language that can be run in pretty much any device following the WORA [5], from desktop operative systems to mobile devices.

This allows us to create a distributed client for this application that will be able to run in an Android or iOs device, making it way easier for all the users to benefit from it without huge limitations.

The GUI is made using the Java swing library [6] to keep dependencies at a minimum, it is a really powerful library that has been used to allow the user to control the desktop application.

The maps image has been taken from a screen capture of Google Maps, the general area of the province of Castelló.

All the communication between nodes (cars and road side units in this scope) are made in a distributed way using the standalone library JADE [7]. JADE is a software framework for the development of applications using the agent paradigm and distributed communication. It complies with the FIPA specification [8] and also comes with a few graphical tools to make the debugging and development of a distributed application easier, since it is a really difficult task due to its nature.

1.5 Structure

Chapter 2

State of the art

This chapter gives an overview of previous work, and how this thesis fits within those studies.

2.1 Smart cities

In the last few years, smart cities have been a hot topic in research [9] given the quantity of available data provided by the internet of things [10] and big data [11] and the newest infrastructures that allow cities to track activities in real time. This leds to the use of this data to improve the quality of life of the citizen, employment[12] and also saving on unused services when they are not necessary.

Smart cities is a general idea of which smart transport is only a small part, but that is the part that this thesis is focusing on.

2.2 Smart transport

Smart transport is a topic that only recently has started to be researched. [13] defines Smart transport as „adequate human–system symbiosis to realize effective, efficient and human-friendly transport of goods and information.“and that is exactly what this thesis is focusing on, a protocol that will make transportation effective and efficient while being human-friendly.

As seen in this review [14] the criteria to find the best alternatives is very varied, it can be based on social, economic, environmental or even land usage. Our protocol is focused

on minimize the network congestion and thus minimizing the total added time of travel times.

Our application helps analysts to test different routing algorithms in a distributed way so, and mix different algorithms within the same simulation. I believe that this tool will be really useful for anyone that wants an initial test since its usage and output are so easily learned.

Chapter 3

Parts of the simulator

This chapter gives an overview of which parts is the simulator composed, how its internal parts work and which communication schemes it uses.

JADE uses a peculiar system of agents and behaviors, agents represent a static class that keeps all the information stored but doesn't do anything by itself, that's the behaviors job. Behaviors are like scheduled actions for that specific class, such as moving the car this time unit or drawing the new added cars in the GUI.

It is important for behaviors not take a lot of time, since that is the way to sharing computing time between resources, you execute your behavior and *go to sleep* while other behaviors are executed. This makes for a really fast and efficient way of sharing resources if used correctly.

In this simulator, every behavior takes on Tick (it represent a second on the simulator timescale) to finish, then depending on its nature it will be repeated until an ending condition is reached.

3.1 Simulator files overview

In this section an overview of what files and classes that are needed for the simulator as well as a brief overview of the classes will be done.

3.1.1 Agents

This subsection details the JADE agents used by the simulator.

3.1.1.1 Car agent

This agent represents a mobile vehicle with a set speed that moves from a starting point to a destination via valid paths. The route that follows is determined by the type of routing algorithm chosen.

3.1.1.2 Event manager agent

This agent reads all the events from a file and executes them at specific points in time, this helps the application to have a deterministic behavior if anyone wants to redo the experiments.

3.1.1.3 Interface agent

This agent is keeps all the information about the GUI and is the one that has to keep all its information updated. It also reads the user inputs (such as changing the timescale).

3.1.1.4 Segment agent

This agent represents a road side unit on one segment of the network, that is the connection between two intersections. The road side unit covers the full extent of the segment. All car agents register on it when entering and deregister when leaving, he is in charge of communicating the GUI the position of its car agents, so that the number of messages is reduced (from one message per car to one message per segment).

3.1.1.5 Time keeper agent

This agent keeps the simulation synchronized between all agents, this makes keeping track of time and making scheduled events possible. This is the key to make all the cars move at the same time and, schedule events at a certain moment of the day (such as 12:00).

3.1.2 Behaviors

This subsection details the behaviors used by the simulator.

3.1.2.1 Car behavior

This behavior is used by the Car agent and calculates the next graphical position of the car. It also registers and deregisters the car from the segments. Registering is done when the car enters in a new segment and deregistering is done when the car exits a segment.

In every tick it moves and registers or deregisters of the segment if needed, this is a cyclic behavior that ends when the car reaches its destination.

3.1.2.2 Event manager behavior

This behavior is used by the Event manager agent. It executes the events that has in memory that have previously been read by its agent, if it is the time to execute that event it sends the instructions to the interested party. It also formats the time displayed in the GUI so that it is human readable.

In every tick it sends all the necessary messages if a event is executes, it also sends a message to update the time. This is a cyclic behavior that ends when the simulation ends.

3.1.2.3 Interface add car behavior

This behavior is used by the Interface agent, it receives the instructions to create a new car (from the manager behavior) and creates the graphical representation of it and keeps it updated.

In every tick it checks if a new car has to be added. This is a cyclic behavior that ends when the simulation ends.

3.1.2.4 Interface draw behavior

This behavior is used by the Interface agent (yes, agents can have more than one behavior) and it updates all the parts of the GUI. It updates all the cars position on the GUI, adds new cars to the GUI, deletes cars that have finished from the GUI and updates the time and the number of cars on the GUI.

In every tick it checks whether he has to update the any part of the GUI or not (usually it has to update all the cars position since they move every tick). This is a cyclic behavior that ends when the simulation ends.

3.1.2.5 Segment listen behavior

This behavior is used by the Segment agent and listens to messages from cars to register or deregister, or to update their position. It also listens for messages from the Event manager in case it needs to change its service level.

In every tick it checks for messages from cars or events and updates its lists of registered cars. This is a cyclic behavior that ends when the simulation ends.

3.1.2.6 Segment send to draw behavior

This behavior is used by the Segment agent and sends the information about the cars that are registered on it to the Interface agent so it updates the car positions.

In every tick if there are any cars on it, it sends their information to the interface agent. This is a cyclic behavior that ends when the simulation ends.

3.1.3 Environmental classes

This subsection describes the classes that the simulator uses to model the road network.

3.1.3.1 Intersection

This class represents an intersection, that is a point on the map where one or more segments end or start. It is only possible for a car to change between segments here.

3.1.3.2 Segment

This class represents the connection between two intersections. Each segment has a origin and an end, it also keeps all the important information such as maximum speed, number of tracks and density. This class also has a list of steps that contain the information for its graphical representation.

3.1.3.3 Step

This class is used to represent a line in the canvas where the graphical map is drawn, it contains the x and y coordinates for the initial and ending point of that line, segments are usually made up of more than one step so its graphical representation is real life like.

3.1.3.4 Path

This class is used to keep the segments and steps that a car has to follow to get from its origin to its destination, this path can be dynamically computed depending on the algorithm of choice.

3.1.3.5 Map

This class represents the whole road network, it reads the data from file and instances all the necessary classes to build the directed graph, it also instances the segment agents. It provides many methods to work with the graph.

3.1.4 CanvasWorld

This class contains all the information that is needed to draw the GUI, it also contains an API that allows the Interface agent to modify its contents.

3.1.5 Main

This is the class that is called when the program is started it spawns the required JADE services. It also provides a few configuration options that are worth mentioning:

- **tickLength**: This parameter is used when the application is used in headless mode, that is without spawning the GUI, which is very useful for servers. This overwrites the default tickLength that is usually set by the user via a slider in the GUI. *Default*: 1L
- **startingTick**: This is the tick at which the application starts, because this tick represents one simulation second you can do something like $7 * 3600 + 30 * 60$ to make it start at 7:30 am. *Default*: $7 * 3600 + 59 * 60$
- **finishingTick**: This is the tick at which the simulation will end. *Default*: $24*3600$
- **numberOfCars**: This parameter allows the user to put a certain number of smartcars at the beginning of the simulation, this is very useful for stress tests. *Default*: 0
- **drawGUI**: This parameter allows running the application in headless mode. *Default*: true

- **startRMA**: This parameter allows the control of starting the JADE Remote Agent Management, which is an interface to an administration panel for the agents. *Default*: false
- **segmentLogging**: This parameter allows the control of the logging system. *Default*: false
- **loggingDirectory**: This parameter allows us to change where the log files will be stored. *Default*: ""

3.1.6 Search algorithms

At the moment of writing this thesis, there are only three implemented algorithms. The simulator uses the Factory method programming pattern [15] so that if any researcher wants to add a new algorithm he or she can do it easily.

3.1.6.1 Shortest path algorithm

This algorithms is an implementation of the Dijkstra's algorithm [16] that looks for the shortest path on the graph. Cars using this algorithm always take the same path given the same origin and the same destination.

3.1.6.2 Fastest path algorithm

This algorithm is a modification of Dijkstra's that focuses on time rather than on distance, this algorithm searches for the longest time for a given segment knowing the maximum speed of the car and of that segment. This **does not** take into account the state of the traffic.

3.1.6.3 Smartest path algorithm

This algorithm is yet another modification of Dijkstra's algorithm, this algorithm focuses on minimizing the trip time and takes into account the current traffic, so that if there is a congestion on a path and the average speed is slower, it will change to a longer path whose total trip time is lower.

3.1.7 Static files

This subsection will describe the static files that the simulator needs in order to run.

3.1.7.1 Events

A file of events is needed so the simulator can run them, the file should be in *csv* format.

Type	Time	Origin	Ending	Maximum speed	Algorithm
newCar	08:53	I-CV10-08	I-N340-07	86	fastest
newCar	22:27	I-AP7-01	I-CV10-03	114	shortest
newCar	11:07	I-AP7-01	I-CV10-04	86	shortest
newCar	11:46	I-CV10-04	I-CS22-04	105	smartest

TABLE 3.1: A few rows of the events.csv file

A Python script to generate this file is also provided in the root of the project, called *generateRandomEvents.py* which allows for an easy creation of test files.

3.1.7.2 Map files

This are the files where the Map class takes its data:

- **intersections**: A JSON file containing the information about the intersections and their position on the graphical map.
- **segments**: A JSON file containing the information of the segments, including origin and destination, maximum speed, length, etc.
- **steps**: A JSON file that contains the steps that make up a segment, remember that the steps are the graphical representation of the segment, whereas the segment does not have a representation by itself.

3.1.7.3 images

There are also three images that are needed on this simulator:

- **icon.png**: That's the icon that will appear on the top left corner of the GUI.
- **legend.png**: This is a legend of the simulator, it helps interpreting the data.
- **red.png**: This is the image used to represent the road network, a Google Maps screenshot.

3.2 Communication ontology

Due to the distributed nature of this simulator, all the communication between classes has to be done through messages. JADE allows us to set different kind of communication ontologies so classes know what type of arguments to expect and what to do with those arguments. We can see in 3.1 the different communication ontologies used between classes, this simulator uses quite a few. A brief description of them is done in 3.2.

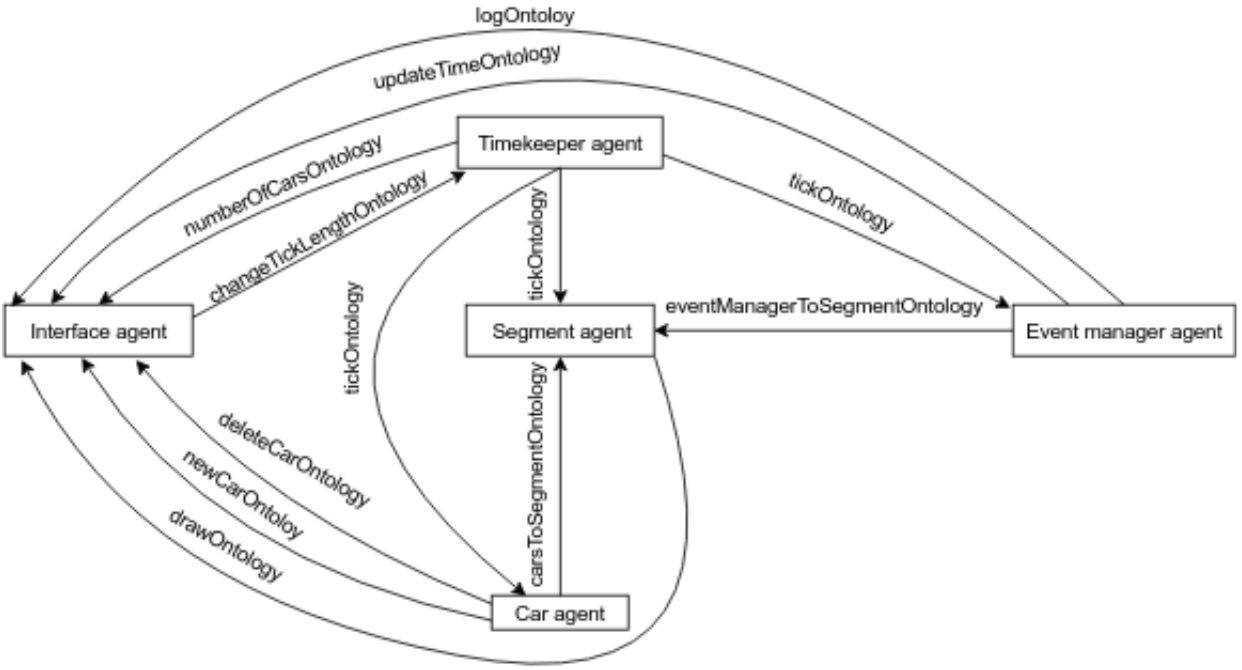

FIGURE 3.1: Ontology communications scheme

Name	**From**	**To**	**Arguments**	**Description**
logOntology	Event manager agent	Interface agent	String	Tells the interface agent which text to add to the log panel
drawOntology	Segment agent	Interface agent	String	Sends the information about the cars that have to be drawn
newCarOntology	Car agent	Interface agent	String	Adds a car for the first time
changeTickLengthOntology	Interface agent	Timekeeper agent	Integer	Changes the ticklength on the simulation
tickOntology	Timekeeper agent	Car, segment and Manager agents	Long	Tells everyone to compute an additional tick
numberOfCarsOntology	Timekeeper agent	Interface agent	Integer	Tells the interface how many cars are running in the simulation
carToSegmentOntology	Car agent	Segment agent	String	Tells the segment to register, deregister or update this car
deleteCarOntology	Car agent	Interface agent	String	Tells the interface to delete that car
updateTimeOntology	Event manager agent	Interface agent	String	Tells the interface to update the displayed clock
eventManagerToSegmentOntology	Event manager agent	Segment agent	String	Tells a segment to change its service level

TABLE 3.2: The complete list of ontologies

Chapter 4

Routing algorithms

In this chapter, well known routing algorithms are proposed and studied.

4.1 What is a routing algorithm

A routing algorithm is a deterministic program that finds a path between an origin and a destination. Usually the network where the path has to be extracted from is a graph, directed or undirected.

A basic graph has two main components, vertices and edges. In the context of this thesis, the vertices are traffic intersections and the edges are the roads that connect them.

Edges usually come with a weight, it is a value that represents the cost of traversing that edge, this could be anything from kilometers to time. Graphs can even have more than one kind of weight depending on what we intend to optimize.

In the context of this thesis, the roads have their length and the maximum allowed speed, this allows us to use the most simple metrics on traffic, time and distance.

A routing algorithm takes advantage of all this information and finds the minimum path depending on the metric we want to optimize, in our case time or distance.

In this thesis we are going to focus on the most simple yet effective path finding algorithms.

4.1.1 Dijkstra's algorithm

Dijskstra's algorithm is a well known iterative algorithm that finds the shortest path between two nodes in its most simple version. This is the most famous path finding algorithm and has been deeply studied [17] [18].

Basically, this algorithm keeps a set with all the open nodes, in each iteration the node with the shortest distance to the starting path is studied, this node is removed from the set and all its neighbors are added to it if they are still to be studied. This process ends when we arrive to the desired node.

4.1.2 Bellman-Ford algorithm

Bellman-Ford's algorithm is a well known variation of Dijkstra's algorithm that is worth mentioning, the main difference between those two algorithms is that Dijkstra cannot work with negative edges while Bellman-Ford can.

This feature is not interesting for this thesis so we will not go into further detail.

4.1.3 A* algorithm

A* is a variation of Dijkstra's algorithm commonly used in videogames. It uses a heuristic to estimate if the node to study looks promising, cutting the complexity time. This algorithm heavily relies on the heuristic function and does not guarantee the optimal path, that's why this algorithm has not been chosen.

4.2 Chosen algorithms

As detailed in the previous section, some algorithms have been studied to find the one that fits better in this simulator, I have chosen Dijkstra's because it always returns the optimal path, you can chose what to optimize (for example time or distance) and it can be easily modified.

4.2.1 Shortest path algorithm

The shortest path algorithm used in this simulator is the most basic implementation of Dijkstra's, knowing the distance between all the intersections the car finds its path looking for the one that takes less distance for it.

This method is deterministic, any car wanting to go from point A to point B will have the same path. Given the dynamic nature of the traffic, this will lead to significant traffic jams if the chosen path crosses a segment that congests very easily.

4.2.2 Fastest path algorithm

The fastest path algorithm is a slight modification of the previous algorithm, it only differs on searching for the minimum time instead of the minimum distance.

For each segment the maximum speed a car can drive on it is

$$Speed_{max} = min(\text{max speed on that segment}|\text{max speed of the car})$$

Because the maximum speed of each car is different, different paths from A to B can be found, which makes this algorithm slightly better to avoid traffic jams. However, it does so by accident and many cars will chose the same path.

4.2.3 Smart path algorithm

The smart path algorithm takes into account the state of the traffic in real time and avoids congested paths. This is the proposed algorithm and it is expected to behave much better than the others.

Again, this algorithm is a modification of Dijkstra's algorithm that minimizes the time that takes the car from going to point A to B. This algorithm is very similar to the previous one, but it takes into account the *current* maximum speed of that segment such that

$$Speed_{max} = min(\text{current max speed on that segment}|\text{max speed of the car})$$

so if a segment is congested the car simply avoids it. Because segments are dynamically congested, this algorithm recalculates its path at every intersection, given that you cannot turn around inside a segment this algorithm effectively keeps you in the fastest path at every moment.

Bibliography

[1] Xue Yang, Jie Liu, Feng Zhao, and N.h. Vaidya. A vehicle-to-vehicle communication protocol for cooperative collision warning. *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004.* doi: 10.1109/mobiq.2004.1331717.

[2] Shane Tuohy, Martin Glavin, Ciaran Hughes, Edward Jones, Mohan Trivedi, and Liam Kilmartin. Intra-vehicle networks: A review. *IEEE Trans. Intell. Transport. Syst. IEEE Transactions on Intelligent Transportation Systems*, 16(2):534–545, 2015. doi: 10.1109/tits.2014.2320605.

[3] Noam Nisan. *Algorithmic game theory*. Cambridge University Press, 2007.

[4] By Java. Java software. URL https://www.oracle.com/java/index.html.

[5] Write once, run anywhere, . URL https://en.wikipedia.org/wiki/write_once,_run_anywhere.

[6] Package javax.swing. URL https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html.

[7] URL http://jade.tilab.com/.

[8] URL http://www.fipa.org/.

[9] Andrea Caragliu, Chiara Del Bo, and Peter Nijkamp. Smart cities in europe. *Journal of Urban Technology*, 18(2):65–82, 2011. doi: 10.1080/10630732.2011.601117.

[10] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1): 22–32, 2014. doi: 10.1109/jiot.2014.2306328.

[11] Anthony M. Townsend. *Smart cities: big data, civic hackers, and the quest for a new utopia*. W.W. Norton and Company, 2013.

[12] Jesse Shapiro. Smart cities: Quality of life, productivity, and the growth effects of human capital. *MIT Press journals*, 2005. doi: 10.3386/w11615.

[13] Dick Lenior, Wiel Janssen, Mark Neerincx, and Kirsten Schreibers. Human-factors engineering for smart transport: Decision support for car drivers and train traffic controllers. *Applied Ergonomics*, 37(4):479–490, 2006. doi: 10.1016/j.apergo.2006.04.021.

[14] Johanna Camargo Pérez, Martha Helena Carrillo, and Jairo R. Montoya-Torres. Multi-criteria approaches for urban passenger transport systems: a literature review. *Annals of Operations Research Ann Oper Res*, 226(1):69–87, 2014. doi: 10.1007/s10479-014-1681-8.

[15] Factory method pattern, . URL https://en.wikipedia.org/wiki/factory_method_pattern.

[16] Dijkstra's algorithm, . URL https://en.wikipedia.org/wiki/dijkstra's_algorithm.

[17] Muhammad Adeel Javaid. Understanding dijkstra algorithm. *SSRN Electronic Journal*. doi: 10.2139/ssrn.2340905.

[18] Steven S. Skiena. Weighted graph algorithms. *The Algorithm Design Manual*, page 191–229, 2012. doi: 10.1007/978-1-84800-070-4_6.