A* BASED ROUTE PLANNING USING REAL-TIME ROAD TRAFFIC DENSITY DATA: A MULTI-AGENT SIMULATION

A MASTER'S THESIS

in

Computer Engineering

Atılım University

by

İLTER KAĞAN ÖCAL

JULY 2010

A* BASED ROUTE PLANNING USING REAL-TIME ROAD TRAFFIC DENSITY DATA: A MULTI-AGENT SIMULATION

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

ATILIM UNIVERSITY

BY

İLTER KAĞAN ÖCAL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR

THE DEGREE OF MASTER OF SCIENCE

IN

DEPARTMENT OF COMPUTER ENGINEERING

JULY 2010

Approval of the Graduate School of Natural and Applied Sciences, Atılım University.

_____

Prof. Dr K. İbrahim Akman

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. K. İbrahim Akman

Head of Department

This is to certify that we have read the thesis "A* Based Route Planning Using Real-Time Road Traffic Density Data: A Multi-Agent Simulation" submitted by "İlter Kağan Öcal" and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____	_____
Asst. Prof. Dr. Hürevren Kılıç	Asst. Prof. Dr. Fatma Cemile Serçe
Co-Supervisor	Supervisor

Examining Committee Members

Prof.Dr.Ali Yazıcı	_____
Assoc.Prof.Dr. Ferda Nur Alpaslan	_____
Asst. Prof. Dr. Fatma Cemile Serçe	_____
Asst. Prof. Dr. Hürevren Kılıç	_____
Inst. Aylin Akça Okan	_____

Date: 09.07.2010

I declare and guarantee that all data, knowledge and information in this document has been obtained, processed and presented in accordance with academic rules and ethical conduct. Based on these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

İlter Kağan Öcal

ABSTRACT

A* BASED ROUTE PLANNING USING REAL-TIME ROAD TRAFFIC DENSITY DATA: A MULTI-AGENT SIMULATION

Öcal, İlter Kağan

M.S., Department of Computer Engineering

Supervisor: Asst. Prof. Dr. Fatma Cemile Serçe

Co-Supervisor: Asst. Prof. Dr. Hürevren Kılıç

July 2010, (61) pages

A major characteristic of real world traffic is the amount of complexity. The vehicles do not travel in organized clusters, but they move in unpredictable ways, with different speeds even when they are on the same route. It is a challenging problem to find the fastest routes in such complex environment. There are navigation devices and software on the market that are capable of calculating the shortest route from a given source location to a desired destination location using traffic information such as maximum speed limits of the roads, historical road database, broadcasting technology over FM band, or similar. However these techniques do not provide real-time information. Therefore, the shortest routes suggested from one point to another may not be the fastest one in real traffic condition at one time. In this study, a multi-agent simulation has been designed and developed to collect and disseminate real-time traffic information. Each vehicle in this environment has been modeled as an intelligent agent. This software simulates the movement of the vehicle on a road map based on real-time data. In the scope of this study, several tests have been conducted in order to analyze the effectiveness of routing based on real-time data. The experiments showed that the proposed real-time data based route planning is both efficient and effective at finding fastest routes.

Keywords: Route Planning, Real-time Route Planning, Multi-Agent Systems, Traffic Information Collection and Dissemination, Agent-based Simulation

ÖZ

GERÇEK ZAMANLI TRAFİK YOL DURUMU BİLGİSİ İLE A* TABANLI ROTA PLANLAMASI: ÇOK ERKİNLİ BİR BENZETİM

Öcal, İlter Kağan

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Yrd. Doç. Dr. Fatma Cemile Serçe

Ortak Tez Yöneticisi: Yrd. Doç. Dr. Hürevren Kılıç

Temmuz 2010, (61) sayfa

Araç trafiğinin önemli özelliklerinden birisi karmaşıklığıdır. Trafikteki araçlar tahmin edilemeyecek şekilde hareket ettikleri için, trafik ortamında düzenli bir araç akışı yoktur. Bu gibi karmaşık bir ortamda, iki nokta arasındaki en hızlı rotanın bulunması zor bir problemdir. Piyasada iki nokta arasındaki en kısa rotayı ve hız limitleri, geçmiş yol istatistikleri, fm bandı üzerinden yayın ve benzeri kaynaklardan alınan bilgiye dayanarak en hızlı rotayı hesaplayan navigasyon cihazları bulunmaktadır. Fakat hızlı rota hesabında kullanılan bu veriler yolların gerçek zamanlı hız durumlarını içermemektedir. Bu bilgiler kullanılarak yapılan hesaplamalarda, o anda kullanılabilecek en hızlı rota bulunamamaktadır. Bu çalışmada, gerçek zamanlı trafik bilgisi paylaşımı ve dağıtımı yapmak için tasarlanmış çok erkinli bir simülasyon yazılımı geliştirilmiştir. Yazılımda, araçların harita üzerindeki hareketlerinin gerçek zamanlı veriye dayanarak benzetimi yapılmıştır. Erkinlerin yaptığı rota planlamasının kalitesinin sınanması için, bir çok aracın hareket ettiği bir ortam oluşturulmuştur. Yapılan deneylerde, önerilen gerçek zamanlı veriye dayalı rota planlamasının, en hızlı rotayı bulmakta verimli ve etkili olduğu görülmüştür.

Anahtar Kelimeler: Rota planlama, Gerçek zamanlı rota planlama, Çok erkinli sistemler, Trafik veri toplama ve dağıtımı, Erkin-tabanlı benzetim

To My Parents

ACKNOWLEDGMENTS

Firstly, I would like to thank my supervisor Asst. Prof. Dr. Fatma Cemile Serçe for her orientation, guidance and encouragement during my thesis study. I would not be able complete this thesis without her support and contributions. Then I would like to thank my co-supervisor Asst. Prof. Dr. Hürevren Kılıç for his suggestions and comments.

I also would like to thank my instructors who taught me during my undergraduate and graduate education. I became a qualified computer engineer and researcher by their unforgettable efforts.

I want to thank to TÜBİTAK/BİDEB for the scholarship they founded during my master's study. It would be harder to complete this thesis without their valuable support.

Last but not least, I would like to thank my parents and brother for always being with me before and during this study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABLE	Agent Building and Learning Environment
AFS	Actual Flow Speed
AID	Agent Identifier
AMS	Agent Management System
API	Application Programming Interface
DF	Directory Facilitator
DS	Driving Speed
EFS	Expected Flow Speed
FIPA	Foundation for Intelligent Physical Agents
FPS	Frames Per Second
GPS	Global Positioning System
GUI	Graphical User Interface
IDE	Integrated Development Environment
JADE	Foundation for Intelligent Physical Agents
JVM	Java Virtual Machine
PDA	Personal Digital Assistant
POI	Point of Interest
RFID	Radio-Frequency IDentification
WAP	Wireless Application Protocol
XML	Extensible Markup Language

CHAPTER 1

INTRODUCTION

While driving a car, we need to know our position on a map and the traffic conditions along the possible paths from the current location to destination. We can already see our position in a map using the current navigation devices. However, traffic information collection and usage in path finding are critical in the context of traffic management. The traffic information is obtained from a variety of resources such as image detectors, police, radars, navigation systems, and so on.

There are many studies conducted on traffic information collection and path finding on road networks. While some research has focused on centralized architecture for data collection and dissemination, some others offer a distributed architecture for this purpose. Path finding problem has a large body of literature resulting with many different methods and techniques developed and discussed. These different types of architectures and path finding techniques are revised and discussed in the scope of this study.

Apart from the research literature, vehicle navigation systems constitute a big market in which many products and technologies are developed. In-vehicle route guidance systems are being manufactured and used since 1990s and there are many companies manufacturing either portable or embedded vehicle navigation systems. After the 21st century, by the modernization of wireless communication technologies, navigation systems had the ability to use wireless communication. There are products on the market which have wireless communication capability and/or make use of any kind of traffic information in route calculation. Moreover, there are some services and

technologies on the market which are used to broadcast traffic information. These products, services and technologies are briefly introduced and discussed.

In this study, a real-time multi-agent system approach is proposed for the vehicles to collect and distribute information about the traffic conditions they are experiencing. The idea in this study is to consider active vehicles in the traffic as agents, which send traffic reports to a centralized server via wireless internet. A centralized server behaves as an agent which collects data from vehicle agents and estimates average traffic flow speed for each road piece on the map. The vehicles are able to retrieve real-time traffic flow speed of a specific set of road pieces when they wish to plan a route between two points. The purpose of the study is to investigate if route planning based on real-time data is more effective and efficient than route planning based on static and statistical traffic data. In order to achieve this purpose, a multi-agent simulation has been designed and developed. In this multi-agent simulation, virtual vehicles are moving on a road map collecting traffic data and using the data collected from other vehicles. The architecture of the simulation, test cases, test results, findings and applicability of this architecture to real life are explained and discussed.

In chapter 2, studies related with real-time traffic data management, navigation technologies and routing mechanisms are discussed. In chapter 3, technical details of multi-agent simulation are given. Later in chapter 4, test procedure and results of route planning based on real-time data are analyzed. Lastly, discussion, conclusion and future work parts take place in chapter 5.

CHAPTER 2

LITERATURE REVIEW

2.1 Traffic Information Collection and Dissemination in Navigation Tools

As the usage of electronic mobile devices such as smart phones, PDAs (Personal Digital Assistant), portable navigation devices became widespread in the last decade, there have been a lot of online mobile services implemented by the help of wireless communication technologies. Nowadays, almost every popular mobile device comes up with wireless embedded technologies such as WAP, GPRS and now the third generation 3G. Another wireless communication technology is Global Positioning System (GPS), which is simply calculating the global coordinates of a specific point on the earth by the help of satellite signals.

There are many studies and services to offer online traffic information by combining mobile devices with GPS technology. General Motors OnStar [1] is one of the most well-known systems. It is a system that constantly communicates with several satellites orbiting the earth. The system triangulates the car's position, retains the information, and, at the driver's request, transmits the vehicle's location to the OnStar Center via a cellular phone. An advisor at the OnStar center can send assistance if it is called for or use the cellular phone to guide the driver to his or her destination.

The other popular GPS services that are currently being used in mobile devices are Navigation Systems. Navigation systems are able to track the position of itself on a map and come up with a set of built-in route planning algorithms which direct the car

driver to a desired destination by calculating the shortest available path. Current navigation systems suggest the shortest path regardless of the real-time traffic density. Some navigation software considers the historical road density while planning routes but they are not updated frequently and they do not consider real-time information. The addressable problem is that the shortest path suggested by the navigator is not always the fastest path; occasionally the path suggested contains some hot routes in a local area which has a traffic jam and causes a critical waste of time and fuel. It is possible to solve this problem by considering not only the lengths, but also the traffic density and speed range of road pieces in the map while calculating the optimum path.

Considering traffic density in this aspect is not a new concept, thus there are some related works in this area. [2] offers a technique to discover the hot routes based on traffic density by gathering real-time data from RFID transponders located in vehicles. However, the outcome of this work does not offer an algorithm that uses this real-time data. [3] comes up with a adaptive fastest path calculation technique which is based on historical traffic density data. This algorithm uses a statistical historical data to calculate a fastest path which is not aiming to predict the current density. In other words, this algorithm is not able to pay attention to an immediate traffic jam caused by an accident or road construction. In [4], a statistical density prediction is proposed similar to [2] but in this work, statistical data and real-time data are combined together to predict the traffic density of a road segment at a specific time. In practice, cars are transmitting their positions to a centralized server via cell-phones, and server can use this information to infer the future trend of the current traffic to estimate the future traffic situation. Experiments in [4] showed that statistical prediction approach has a high success rate but the density data obtained by this approach is not used for real-time fastest path calculation. Another study [5] has developed a data collection platform named “CarWeb”. All devices running this application are transmitting their speed and position to a centralized server similar to [4] and it is possible to use this platform to display the real-time density map of a road network. However, again this work is not bound to a path finding system which uses this data.

Almost every navigation system on the market has an option that suggests the fastest path rather than the shortest one by considering maximum speed limits of the route segments. However, as we all experience, speed limits of road segments are not reflecting the real traffic flow speed. For example, the rush hour in a highway causes one to go slower than the speed limit; one other narrow street may let one drive faster than that crowded highway. This means that speed limits of road segments are not trustful heuristics for the fastest path calculation. To overcome this problem, real time traffic data distribution methods are being developed by manufacturers.

TomTom navigation software is providing a historical road database which stores the status of road segments on distinct time intervals of a week. TomTom software uses this database for calculating the most possible fastest route at a specific time. This technology is called TomTom IQ Routes ™ [6] (Figure 1).

Figure 1 TomTom IQ Routes

TomTom released a new source of traffic information called HDTraffic [7]. This technology collects anonymous traffic data from millions of mobile phone users in the traffic. Collected data is combined and compiled with the historical database of IQ Routes and traffic incident reports are generated. These reports include the road id having the incident, length of the queue occurred because of this incident, and the delay time aroused. These incident reports are fed to TomTom route planners over internet and route planners use these reports to find the smartest route. More information can be found in the white paper of TomTom HDTraffic [8].

Radio Data Service –RDS- is a data broadcasting technology over FM Band. Movie times, gas prices, traffic incidents, traffic flow, weather reports are some popular subjects of RDS. There are some Traffic RDS providers such as NAVTEQ Traffic [9], MSN Direct [10], and XM NavTraffic [11] which are collecting traffic data from police department reports, local road sensors and compiling traffic flow reports. Then these reports are broadcasted using RDS. Some navigation devices have a built in FM RDS receiver and the navigation software embedded to that device is capable of using that real-time traffic flow reports for route calculation. RDS is a subscription-based service which requires monthly charge. Alpine, Garmin, Pioneer are known brands that have products capable of receiving and using RDS data. Some vehicle manufacturers are placing a RDS compatible navigation system to their products as standard or optional feature. Similar to RDS, there is an alternative and new trend of global data broadcasting technique called Satellite Radio. NAVTEQ and XM are providing service both via RDS and Satellite Radio.

In [12], an adaptive traffic data broadcast scheme is proposed for an Inter Vehicle Communication IVC-based decentralized traffic information system. Proposed scheme uses 802.11 wireless communications which is limited by a range of at most 1000 meters. Each vehicle is broadcasting its estimated traffic condition of the currently experienced road segment. This reduces the possibility of an individual vehicle to learn about the condition of a road which is 10 kilometers away from itself. In order to make it possible, each vehicle should re-broadcast the data received from others which may explode the amount of packets distributed by the vehicles. A time interval of 10 minutes is chosen to update live data which is not actually a real time system. Briefly, a decentralized system limits the range of data sharing distance among the vehicles.

There are already a lot of research conducted on the area of automated traffic management and centralized traffic data acquisition such as traffic prediction patterns [13] and network flow theory [14], sensors developed include sound-based systems [15],[16], laser-based systems [17] and vision-based systems [18]. Trafficopter[19] is one of these studies which uses multi-agent technology similar to this study. It is a multi-agent system designed and developed to collect, process and distribute traffic information in a completely decentralized and distributed fashion.

Each vehicle is an agent. The vehicles themselves collect and distribute information about the traffic conditions they are experiencing to other interested vehicles. Trafficopter uses a communication network built from nodes with dynamic topology. There are message objects moving from node to node. The agents use message objects in order to request and propagate information about traffic. Similar to [12], distributed fashion used in this work constructs an obstacle that may prevent agents to obtain the condition of a further road. Decentralized architecture has an advantage of avoiding single point failures, but this approach only allows message sharing between vehicles close to each other and on similar directions. However, a centralized system allows message sharing between any agents regardless of distance.

In this study, a multi-agent system architecture to solve this optimum route planning problem is proposed. Using a multi-agent approach, each active navigation device in the traffic is considered as an intelligent agent which is able to transmit its positions to a central server agent and receive the momentary traffic density on a specific road piece. Each agent in the system architecture proposed has been designed to use real-time knowledge of the road for optimum route planning.

2.2 Route Finding Algorithms

Path finding problem on road maps is a known topic on which many research have been conducted. Many different graph search and path finding algorithms appeared in the literature. Some of them are aiming to find shortest path in the least possible computation time, some other are focused on finding a acceptable path by pruning the search tree to improve computation time in exchange for finding the shortest path. In addition, there are algorithms specific to topological characteristic of road network. Moreover, there are heuristic search methods which increase the computational efficiency of shortest path searches. In [20] usage of heuristic search methods in vehicle navigation is discussed. [21] examines different heuristic strategies such as limiting the search area, decomposing search problem, limiting the searched links in graph and their combination. In this study, A* search is used to calculate the path between two points on the map.

A* search [22] is a heuristic best-first search algorithm which finds the least-cost path from a given initial node to a goal node. A* search uses a distance plus heuristic cost function f(x) which is sum of two functions g(x) and h(x) where; g(x) is the path cost function, cost from starting node to current node, h(x) is an admissible heuristic of the distance from current node to goal

2.3 Agent Development Environments

There are several Java-based agent development environments such as ABLE[23], AgentBuilder[24], Aglets[25], FIPA-OS[26], JADE[28], JATLite[29], and so on. In this study, JADE was chosen as the agent development environment. The brief descriptions about JADE and other agent development environments are provided below.

ABLE (Agent Building ad Learning Environment)[23] is a Java framework, component library, and productivity tool kit for building intelligent agents using machine learning and reasoning. The ABLE research project is made available by the IBM T. J. Watson Research Center. ABLE provides a set of reusable JavaBean components, called AbleBeans, along with several flexible interconnection methods for combining those components to create software agents. It provides a graphical user interface-based interactive development environment [23]. It is possible to obtain information about ABLE at www.alphaWorks.ibm.com/tect/able.

AgentBuilder [24] is an integrated software toolkit that allows software developers to quickly develop intelligent software agents and agent-based applications. It is developed by Reticular Systems Inc. There are two main products in AgentBuilder toolkit such as Agent Builder Lite and AgentBuilder PRO. The former is suitable for building single-agent standalone applications and small agencies. The latter one has all features of Lite plus an advanced suite of tools for testing and building multi-agent systems[24]. This toolkit uses a high-level, agent-oriented programming language and provides a suite of graphical programming tools for configuring agents and specifying their behaviors. AgentBuilder is intended to enable developers who have no artificial intelligent background to build intelligent applications. In addition to agent-level development and debugging tools, it provides a set of graphical project

management and domain analysis tools[24]. It is possible to find additional information on Agent-Builder's Web site at www.agentbuilder.com.

Aglets [25] are Java objects that can move from one host on the Internet to another. Aglets are developed by IBM. That is, an aglet that executes on one host can suddenly halt execution, dispatch itself to a remote host, and resume execution there. When the aglet moves, it takes along its program code as well as its data. IBM turned over the source to the open source community. IBM provides API documentation and the Aglets software development kit[25]. For more information about aglets, refer to www.trl.ibm.co.jp/aglets.

FIPA-OS [26] is an environment developed at Nortel Networks. There is an agent communication language standard, named Foundation for Intelligent Physical Agents [FIPA] [27]. FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. FIPA, the standards organization for agents and multi-agent systems was officially accepted by the IEEE as its eleventh standards committee on 8 June 2005. FIPA specifications represent a collection of standards which are intended to promote the interoperation of heterogeneous agents and the services that they can represent. FIPA-OS is an open-agent platform that supports communication using FIPA. It provides the set of platform services that are specified in the FIPA agent standards, including an agent management system for life-cycle management, a director facilitator or yellow pages service, and an agent communication channel for FIPA-compliant messaging and interaction protocols. For more information about FIPA-OS, refer to http://fipa-os.sourceforge.net

JADE (Java Agent DEvelopment Framework**)** [28] is a FIPA-compliant java agent development environment developed at CSELT S.p.A in Torino, Italy. JADE is a software framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the [FIPA] specifications [30] and through a set of graphical tools that supports the debugging and deployment phases.

A multi-agent system launched on JADE is called a platform. A platform is composed of agent containers which can be deployed on different computers over the

network. A container is JAVA process which provides the JADE run-time and services needed by agents to live. Agents are hosted and executed in a container. Many agents can be placed in a single container. There are two options in creating a container. If an IP address is given, the container joins to an existing platform which is hosted on the computer having that given IP address. Otherwise, the container becomes a main container, which is the bootstrap point of a new platform. To create a new agent platform, firstly a main container should be launched and then the other containers should register to main container.

When a main container is launched, two agents are automatically created which are Agent Management System [AMS] and Directory Facilitator [DF]. The AMS provides white-page and life-cycle service, maintaining a directory of agent identifiers [AID]. DF is the agent who provides the default yellow page service in the platform [31]. Agents can find the AID of other agents by asking to DF.

A user defined agent is implemented by extending the class "*jade.core.Agent".* An instance of any class derived from this class can be launched as an agent in the container. An agent can be created either when launching the container or after the container is launched. There is a method named "*setup",* which should be overridden after extending the "*Agent*" class. This method acts as a constructor; it is called by the agent right after it is created. Initializing the agent components and parameters can be done in this method. Each agent has a local name in its container, and a global name in the main container. Local name is given by the programmer; global name is automatically given by the main container.

JADE provides a very useful messaging mechanism which enables the programmer to establish communication between the agents over internet. Programmers do not need to have any socket programming knowledge to send messages between the agents through internet. Message objects are created as an instance of "ACLMessage" class in JADE which contains two types of content in it. First one is a pure string and second one is a serializable java object. In addition, each ACLMessage has a performative which can be set by the programmer. Performatives have formal semantics defined by the FIPA specification. Briefly, these performatives are used to distinguish the message types, and then messages are processed according to its semantic by the receivers. Once the object instance is

created, message content and performative is set, AIDs of the receivers are specified and the message is sent by the methods provided by agent interface.

An agent is programmed to perform some set of dependent or independent actions and these actions are defined as behaviors. Usually, these behaviors need to run simultaneously such that, when a behavior is performing a long task, another behavior should perform a different task. Some tasks are performed just for once, some are repeated when it is necessary, some are repeated continuously and some are repeated periodically. Moreover, all concurrent tasks should be able to run independently. That's why agents should be multi-threaded and behaviors should be defined as JAVA threads. Life cycle state and synchronization of these threads should be handled carefully.

JADE comes with a set of abstract behavior classes that can be used to adapt the task types given above. The only thing that programmer needs to do is to choose an appropriate abstract behavior class and extend it by implementing the abstract methods ***action*** and ***done*** which are derived from the abstract class. After a behavior is created and wrapped by a threaded behavior factory and it becomes a threaded behavior that can run simultaneously by other behaviors and independently from other behaviors. All state transitions, blocking and synchronization operations of behavior threads are handled by JADE.

In order to add a behavior to an agent, an instance of a behavior class can be created in *setup* method and added to this agent's behavior pool. All added behaviors start execution immediately after the *setup* method ends.

Some abstract behavior classes which are commonly used in the implementation phase of this thesis are explained below.

OneShotBehaviour is an abstract behavior class which can be extended by the programmer to implement operations that need to be executed just once.

CyclicBehaviour is another abstract behavior class which can be used to implement tasks that are executed forever. This behavior type is mostly used for reactive tasks such as message listening and replying.

TickerBehaviour is also an abstract behavior class which is suitable to implement the operations that need to be repeated periodically in certain time intervals.

CHAPTER 3

A MULTI-AGENT SIMULATION FOR REAL-TIME TRAFFIC INFORMATION COLLECTION AND DISSEMINATION

In this section, design and architecture of the multi-agent simulation for traffic information collection and dissemination using real-time data is explained. This simulation has been designed and developed to analyze if the route planning approach based on real-time data is more effective and efficient to find faster routes than shortest and statistical route planning approaches.

3.1 Overview of the Architecture of the Simulation

Java was chosen as the programming language. There are many reasons for choosing JAVA SE as implementation language for the simulation software. One of the reasons is that there is a powerful, free and ready-to-use agent development framework written in JAVA, named JADE as explained in chapter 2. Both JAVA and JADE have mobile editions which are very similar to the standard editions. This brings the ease of converting and migrating the simulation software to a mobile navigation application without making huge amount of code changes. Second reason is that there are some professional and free Integrated Development Environments such as Eclipse and Net Beans for JAVA. In this thesis, Eclipse is used during the implementation. Another reason is that the powerful collections framework in JAVA. Vectors, hash maps, array lists are frequently used in agent implementations and these objects are sent between the agents. Since these classes in JAVA are

serializable, it is possible to send their instances to other agents over network. Figure 2 shows the general overview of the simulation software.

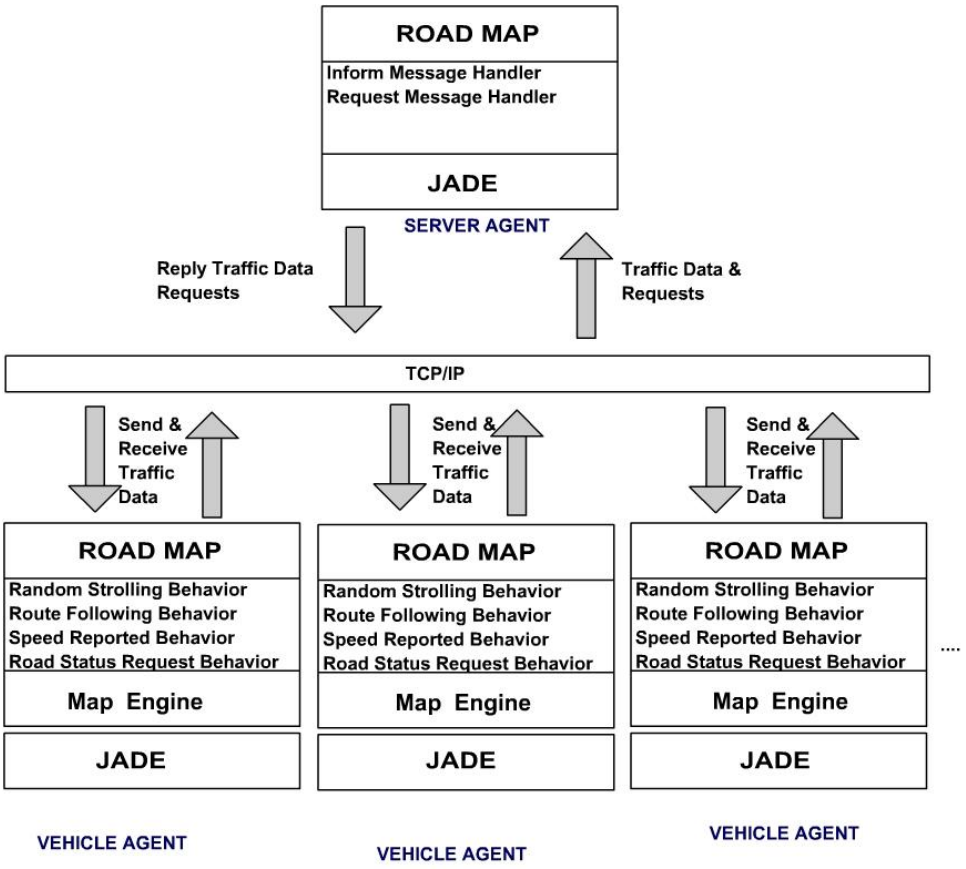


Figure 2 Overview of Architecture of Simulation

The design has five main components:

- Traffic Data
- Road Map
- Vehicles (Agents)
- Server
- Route Planning

Each component is explained in detail below.

3.2 Traffic Data

Traffic data that is aimed to be shared between agents is the speed of the vehicles moving along road segments and average speed of unique road segments. All vehicle agents and server agent assumed to use the same vector-based road map. In this road map, every road segment has a unique id.

The reports that vehicle agents send to the server agent contain the momentary movement speed of the vehicle, and the unique id of the road segment on which the vehicle is moving. These reports also contain the reporting time and the exact coordinates of the vehicle at that moment. These reports are received by the server agent and the average flow speeds of road segments are calculated. As new reports arrive from the vehicles, average flow speeds of the road segments are recalculated by the server agent.

Vehicle agents request traffic data from the server agent by send a list of road ids. Server agent's response to these requests contains the calculated average flow speed of the requested road segments. Vehicles use this information in route calculation algorithm while determining the travel costs between the nodes.

3.3 Road Map

In order to develop the simulation software and test the multi-agent system, it is required to use a road map. For this purpose, a road map storage format is constructed and a sample road map is generated using that format. In Section 3.3.1 the format of the road map is explained in detail. Section 3.3.2 explains from where and how the raw vector data for the sample road map is obtained and how it is processed and modified to use in this work.

3.3.1 Road Map Format

Vector based road map used in this thesis is constructed by nodes, edges and roads. Nodes are start or end points of edges. An edge is a piece of a road which is defined by two nodes. An array of one or more edges constructs a road. Road map is stored in an XML file in XML format. Firstly the node definitions are written in the XML file, and then roads are defined by using those nodes.

```
<node>
        <nid>27046633</nid>
        <lat>39.9151841</lat>
        <lon>32.8245961</lon>
</node>
```

A node is a point on the earth which is defined by a node id, latitude and longitude. Altitude is not taken into account. An example representation of a node in XML file

is given above. The longitude and latitude values in this representation are in World Geodetic System 84 standard [32] which is utilized in Google Earth [33].

Two nodes are not enough to define a curvy road because two points in space can define a straight line segment. A curve can be represented by dividing it into line segments as shown in Figure 3. This is a common way of representing roads in GIS applications. That's why a road in the map is defined by a line string. In the map used in this application, one or more edge elements form a road. Edges are not defined explicitly in XML file format, but they are extracted from road elements and stored in device memory.

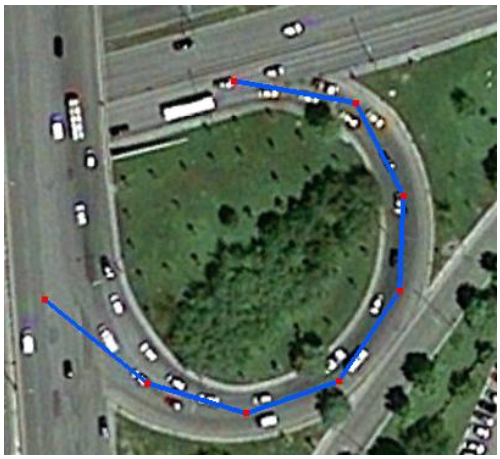


Figure 3: Representation of a curvy road by line segments.

A road is defined by the following elements;

- **Unique road id:** A unique integer that identifies the road.
- **One way element:** Set to true if the road is one way. This means that traffic can only flow in the direction from the first node to last node. Road is recognized as duplex if one way element is omitted or set to false.
- **Type:** Roads are classified depending on their width, lane count, flow speed and popularity. Road class info is stored in this element. Types are discussed later.
- **Name:** Name of the road.
- **Nodes:** An array of node ids that forms this road.

An example representation of a road in XML file is given below.

```
<road>
    <oneway>true</oneway>
    <rid>18758723</rid>
    <type>residential</type>
    <name>17. Sk.</name>
    <nodes>60564387 193640876 404730315 193640882 193640896
277127189 277130490</nodes>
</road>
```

In this example, nodes having the unique ids given in the *nodes* element construct 6 edges. Chain of these edges forms the road.

For each road type there are three values defined such as expected flow speed of road in km/h, width of the road and visibility level. Expected flow speed is used as a base value while simulating the movement of the vehicles and also used in route calculation. Usage of this value is covered later. Width is used to determine thickness of the road when drawing the map on to device screen. Visibility value is used by the map drawing engine to decide whether a road should be drawn to screen or not, depending on the zoom level of the map. Road types, their explanation, expected flow speed, width and visibility level are shown in Table 1.

Table 1 Road types and their properties.

Road Type	Explanation	Expected Flow Speed(EFS)	Width	Visibility
Motorway	Autobahn or intercity highway having 2 or more lanes.	120 km/h	10	10
motorway_link	The link roads leading to/from a motorway from/to a motorway or lower class road	70 km/h	7	10
Trunk	Roads connecting towns. Need not necessarily be a divided highway	90 km/h	8	8
trunk_link	The link roads leading to/from a trunk from/to a trunk or lower class road	60 km/h	6	8

Table 1 Road types and their properties

Road Type	Explanation	EFS	Width	Visibility
Primary	First class main roads in the city, mostly connecting districts	70 km/h	7	7
primary_link	The link roads leading to/from a primary road from/to a primary road or lower class road	50 km/h	5	7
Secondary	Second class main roads inside the districts	60 km/h	6	5
secondary_link	The link roads leading to/from a secondary road from/to a secondary road or lower class road	40 km/h	4	3
Tertiary	Third class roads usually having a rotten surface.	40 km/h	4	2
Residential	Narrow streets accessing or around residential areas.	40 km/h	4	2
living_street	A street where pedestrians have priority over cars, maximum speed is low.	20 km/h	2	2
Unclassified	Any other road whose class is not known or does not match with the ones above	40 km/h	4	3

3.3.2 Road Map Data

In the scope of the study, different road maps are developed for different purposes. An imaginary road is generated and used in routing performance test phase. Detailed information about this road map and reason of using an imaginary map is explained in chapter 4.

Moreover, a real road map of a district in *Ankara* is also used for the validation of the map drawing engine. Satellite snapshot of the covered area taken from Google Earth is shown in Figure 4.

For the selected area, map drawn by Google Maps [34] is shown in Figure 5. Google Maps has an Application Programming Interface which allows users to use this map as a canvas image to share locations, point of interests on it. This API does not allow programmers to access the raw vector data, which is used to draw this map and route calculation. The API only provides the raster image of a specified coordinate at a

given zoom level. Because of this, it is not possible to use Google Maps data in this thesis.

Figure 4 Satellite snapshot of the area where the road map exists in.

Figure 5 Map of area drawn by Google Maps.

The raw vector based road map data used in implementation is taken from OpenStreetMap [35] which is an open source world map project. Content of this project is constituted in a way that is very similar to Wikipedia. Volunteer people from all around the world are drawing and identifying the roads and their types, rivers, lakes, buildings, properties, zones, place marks, borders, POIs, passages, traffic lights, junctions and many other objects on satellite snapshots. All content is protected by "Creative Commons Attribution-ShareAlike 2.0" license [36]. OpenStreetMap allows users to export the map data in XML format use this data under this license.

Figure 6 Exported area from OpenStreetMap

Map of Bahçelievler, which is a district in Ankara, is exported from OpenStreetMap in XML format. The exported area is approximately 6.8 km^2 and shown in Figure 6. Exported XML data contains a huge amount of data including POIs, borders, traffic lights and many other objects which are not necessarily needed in the current implementation. Moreover the XML structure of OpenStreetMap is different from the structure introduced in Section 3.3.1. To eliminate redundant data and convert the

XML structure to our format, a converter program is written in C# .NET. This program reads the XML file exported from OpenStreetMap, parses and loads all elements to memory, eliminates the redundant data and then rewrites the necessary elements to a new XML file in the format used in this study.

Resultant XML file represents a simplified map having only the node and road data. The map exported from OpenStreetMap contains all the roads which intersect the selected rectangle area. Some long roads are defined by more than one piece. Pieces that stands outside of this rectangle are not exported results with some one way roads that end before being connected to another road. To prevent the strolling vehicle agents from stucking in such dead ends during the test process, all nodes which are not connected to a neighbor are detected and pruned from the map or connected to a close node. Finally a map having 289 roads, 1674 edges, 1443 nodes and no dead ends is obtained. Total length of the roads in the map is 94789.76 meters. Size of the final XML file is 213 kb without compression. This XML file is parsed in JAVA by using SAX [37] and the map is drawn on a panel by using 2D Graphics as shown in Figure 7.

Figure 7 Map drawn by using JAVA 2D Graphics.

This road map of “Bahçelievler” is used to validate the correctness of the map drawing engine of developed in this thesis. When drawn map is compared by the original ones in Google Maps and OpenStreetMaps, the positions of roads seem correct, validating the accuracy of the distance, projection, mapping functions implemented inside the map engine.

3.4 Vehicles

Components existing in a vehicle agent and their interactions are shown in Figure 8.

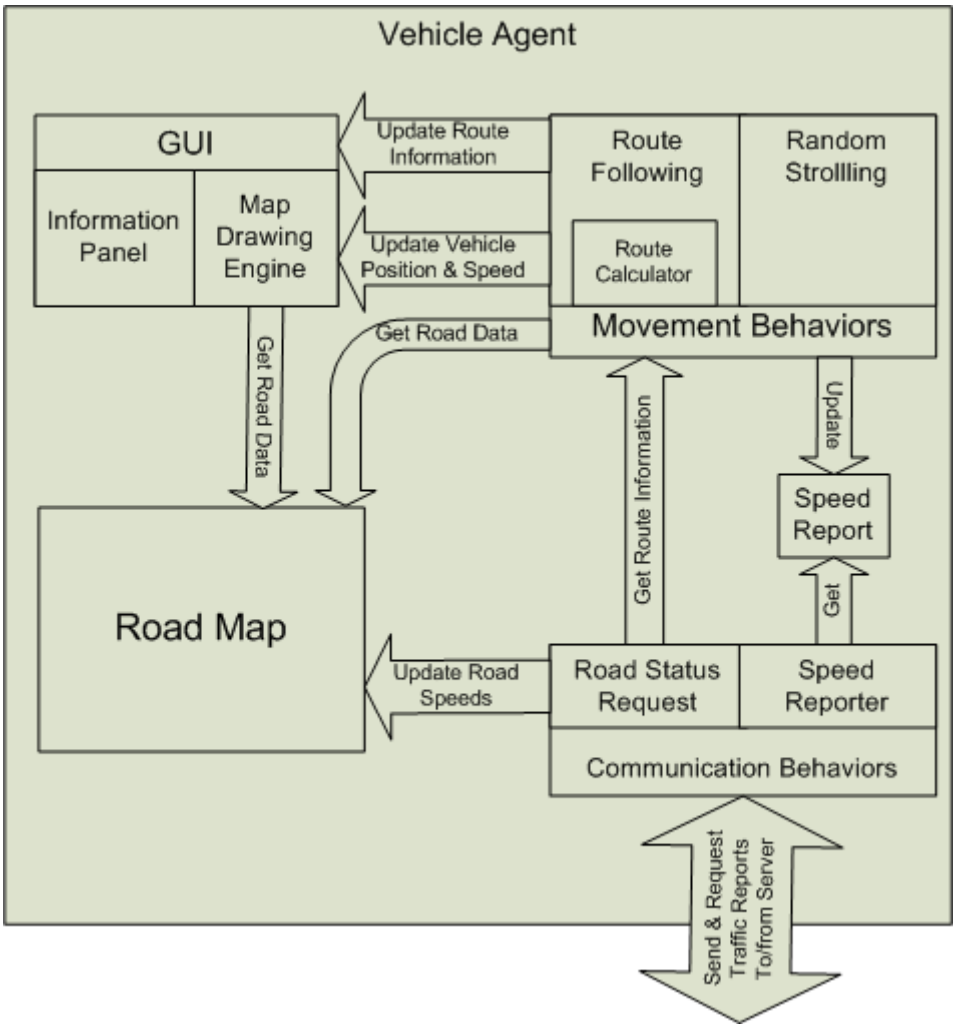


Figure 8 Internal structure of vehicle agent

As it is mentioned in Section 3.2, all vehicle agents and server agent are using the same road map file. Road map is at the bottom level of vehicle agent architecture. Every vehicle agent has a map engine which draws the map on a panel and marks the vehicle’s momentary location. This map engine also draws the active route on the

map. Map drawing engine is embedded into a GUI which shows the active route details, speed info and name of the edge on which the vehicle is moving. There is a class named `Route` which finds and stores the route between two points by using the A* search algorithm. When a route is needed by the vehicle agent, an instance of route class is generated and used. There exist four different behaviors in a vehicle agent as explained in Section 3.4.2.

There are five arguments required while creating an agent and they are explained below;

- *Showgui* is an argument which may take value true or false. Depending on this argument GUI window of the agent is shown or not. GUI is explained in detail in later sections.
- *agentType* is an argument that determines the type of the agent. If *agentType* is 1, agent is a randomly strolling agent. If this argument is 2, agent becomes a routed vehicle agent.
- *startNodeId* is an argument to place the agent at a specific node. The value must be a valid node id in the map. If this argument is -1, agent starts its journey from a randomly selected node.
- *destNodeId* argument is used to set a destination for the agent. The value must be a valid node id in the map. If this argument is -1, agent selects a random node as destination. This argument is ignored when *agentType* is 1 because randomly strolling agents do not have a destination.
- *routeMode* argument is necessary for routed agents and states the routing mode used by the agent. If *routeMode* is 1, agent follows the shortest route to go to destination node from start node. If route mode is 2, agent follows the fastest route. If route mode is 3, agent follows the real time fastest route.

When a vehicle agent is created, setup method firstly parses the arguments explained above, initializes its internal variables. Then it searches for the server agent by using the Directory Facilitator provided by the main container. When the AID of the server agent is found, it is stored for later communication. Finally, the behaviors are defined, created and agent becomes ready to run.

3.4.1 Vehicle Agent Types

There are two types of vehicle agents available in this scenario such as Randomly Strolling Vehicle Agents and Routed Vehicle Agents. These agents are classified according to agent's movement character.

Randomly Strolling Vehicle Agent: The agent starts from a node, and goes to one of its neighbors randomly. After arriving to the next node, another random neighbor is selected and agent starts to move there. Agent repeats this behavior as long as it is active in the system. The aim of placing these agents on the system is collecting/initializing the traffic data. They are strolling randomly on the map and reporting their position and speed periodically. They do not perform any route calculation or route following. These agents are shortly called figure actors.

Routed Vehicle Agent: The agent starts from a node and moves to a destination node selected. This agent firstly calculates the route between these two nodes and follows that route until it reaches to destination node. The aim of placing these agents on the system is to test the performance and see the benefits of real time traffic data usage in route calculation.

3.4.2 Vehicle Agent Behaviors

Behavior types existing in JADE are explained in chapter 2. In a vehicle agent, there are four different behaviors implemented and they are explained below.

Random Strolling Behavior is a cyclic behavior which simulates the random movement of vehicle on the map. If the start node is not given, a starting node is randomly selected. Once the start node is known, one of its neighbor nodes is selected randomly and vehicle starts moving to that location. After vehicle arrives to the new node, this node is selected as the new start node and the behavior goes into a cycle, repeating the process above forever. This behavior is activated if *agentType* is 1. Figure 9 depicts the flow diagram for this behavior.

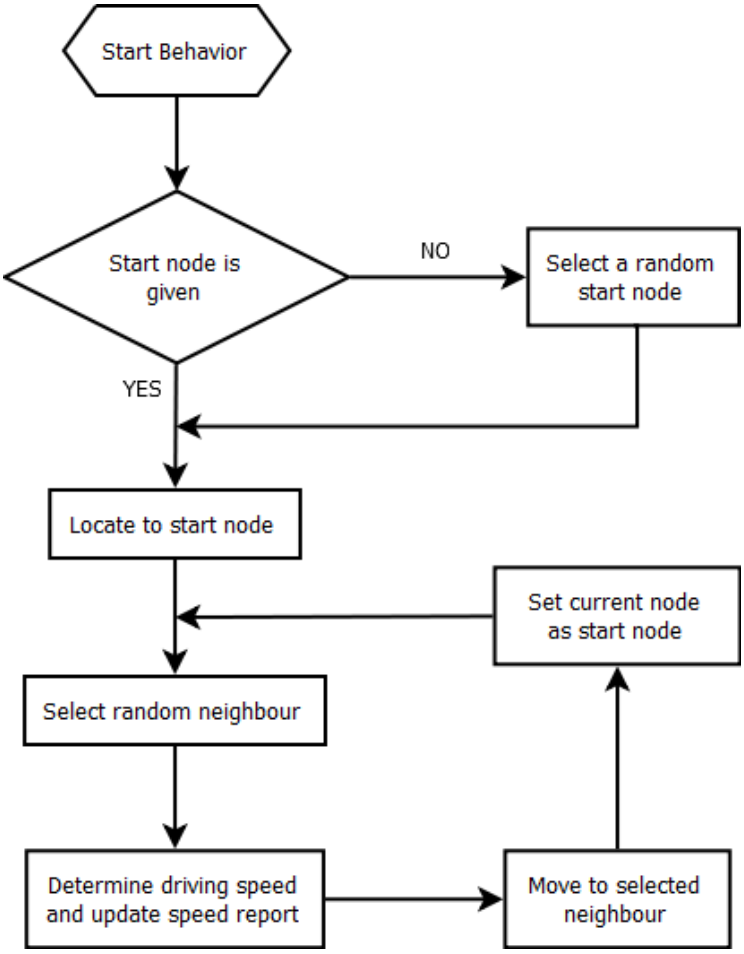


Figure 9 Flow Diagram of Random Strolling Behavior

Route Following Behavior is a ticker behavior and simulates the movement of the vehicle between a start node and a destination node. If the start node is not given, a starting node is randomly selected. Similarly if the destination node is not given, a node is randomly selected. Then, route is calculated between these two nodes. Vehicle follows this route until it reaches to destination node. If the destination node is not given when launching the agent, and when vehicle arrives to destination, this node becomes the new start node and selects a new destination randomly. This process is repeated on each tick. If the destination is node given, agent stops moving when it arrives to destination. It writes the route info into a log file and terminates itself. More information about log file is given in Section 3.9. The flow of the states in behavior is shown in Figure 10.

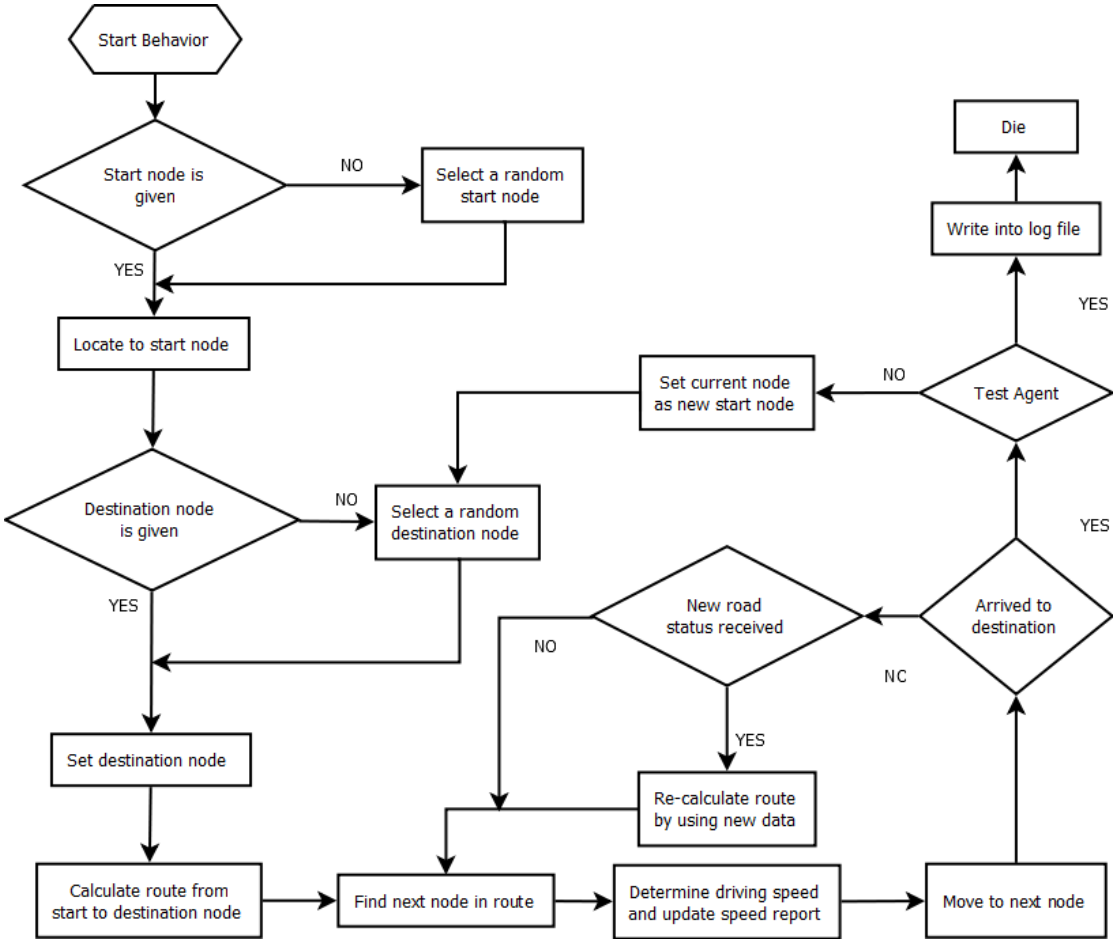


Figure 10 Flow Diagram of Route Following Behavior

Speed Reported Behavior is another ticker behavior which is responsible from speed reporting. As it is mentioned in previous sections, vehicles are reporting their position and speed to the server agent periodically. In each tick, a speed report object containing the edge id, vehicle id, report time and driving speed is generated. This object is wrapped in an ACLMessage object who's performative is "INFORM". Finally this ACLMEssage is sent to server agent. Reporting period length can vary. In tests, reporting period is 10 seconds. This behavior and one of the movement simulation behaviors run simultaneously. On every movement, movement behavior updates the speed report object, so that speed reporter behavior reports the latest situation of the vehicle. Figure 11 shows the flow diagram of Speed Reported Behavior.

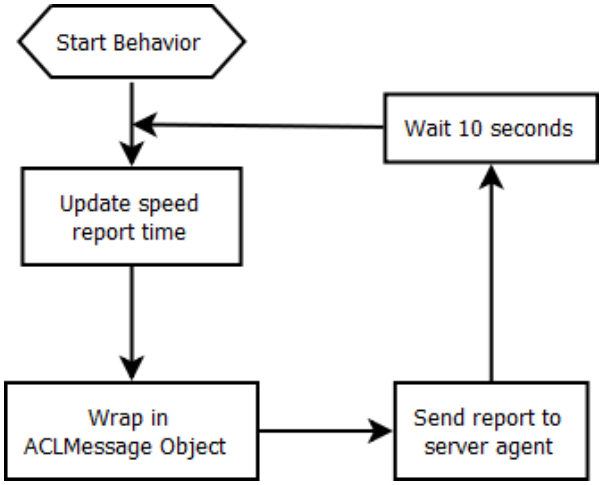


Figure 11 Figure of Speed Reported Behavior

Road Status Request Behavior is the last behavior of a vehicle agent. This is also a ticker behavior which requests the flow speed of all edges in a circular area from the server agent. This circular area is determined in two different ways. If the agent is a randomly strolling agent, center of the circle is the current position of the vehicle, and the radius is 2000 meters. If the agent is a routed agent, center of the circle is the midpoint of start and destination nodes; radius is the distance between start and destination node. For routed agents, this circular area contains a larger set of edges whose speed info is necessary in real time route calculation. After the circular is determined, the ids of all edges that are inside this circular area are found, and these ids are stored in an "ArrayList" object. This object is wrapped in an ACLMessage object who's performative is "REQUEST". Finally this ACLMessage is sent to server agent. Then this behavior waits until server sends a reply. When reply is arrived from server agent, speed data of all requested edges are updated. If vehicle agent is a routed agent using real-time routing, its route following behavior is warned about this update. When a new road status update is done, route following behavior re-calculates its route. By this method it tries to avoid any newly slow downed roads if possible. In testing phase, this request and update operation is repeated in every 30 seconds. Figure 12 depicts the flow of events in this behavior.

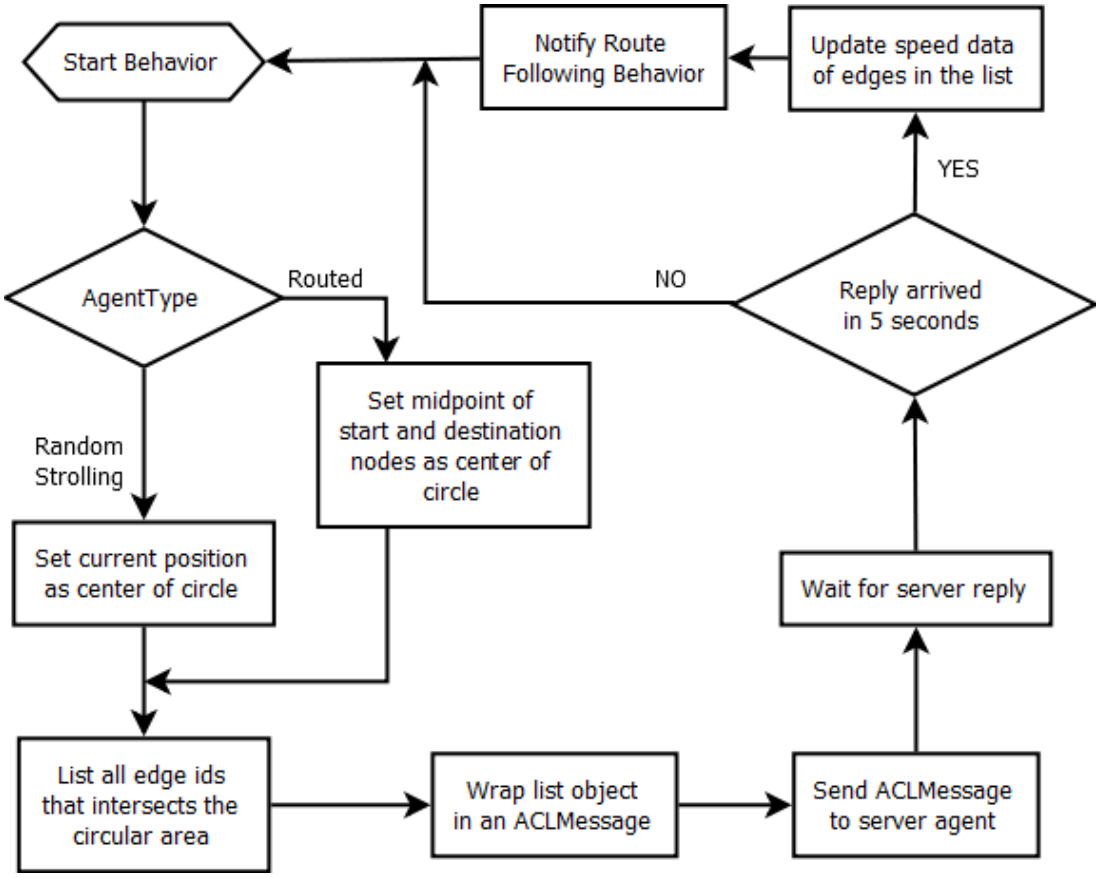


Figure 12 Figure of Road Status Request Behavior

3.5 A Vehicle's Move

Firstly a vehicle is placed on a node, meaning that vehicle is at the exact coordinates of that node. Then vehicle starts moving to another node. This movement is done through an edge, which is defined by two distinct nodes and it is a straight line. To simulate the movement between these two points, firstly a driving speed for that edge is determined. This driving speed is estimated by a base value and a random variance value. Unit of speed is kilometers per hour (km/h).

$$V_{driving} = V_{base} + r$$

V_{base} is the latest flow speed received for that edge from the server. If no agents reported speed for that edge before, server will reply with the predefined expected flow speed. Random variance r is a decimal value between -10 and 10.

For example, a primary type edge "e1" has expected flow speed 70 km/h. The first vehicle passing through e1 will have a random speed between 60km/h and 80km/h. Let's assume that first vehicle agent "VA_1" passing through e1 had determined its

random speed as 67.7 km/h and reported this speed to the server. After that another vehicle agent "VA_2" requested speed report for a circular area containing e1 and reply received from server says that speed for that road is 67.7 km/h. This value is the initial value for speed calculation of VA_2 for e1. Speed of VA_2 will have a value between 57.7 km/h and 77.7 km/h. Let's say VA2 driven through e1 with 63.4 km/h speed and reported this speed to server. New average speed data for e1 on the server side will be (67.7 + 63.4)/2 which is 65.55. As vehicles pass from e1, this data will vary and may result either with a slow flow speed or with a fast flow speed.

As it is seen from the example scenario above, random speed estimations of vehicles will affect each other. By this method at some roads in the roadmap a fluent traffic may occur; at some other roads a traffic jam may occur. The non-deterministic character of a traffic flow environment has been tried to be simulated in this way.

After the driving speed though an edge is calculated, next step is moving the vehicle on this edge with that speed. There are two different movement simulation types. If the vehicle's GUI is disabled, there is no need to show the movement of the vehicle on the edge visually. Simply the length of the edge is divided into the speed and the duration required for the vehicle to drive through that edge is calculated.

$$T_{edge} = L_{edge} / V_{driving}$$

Vehicle is located in the first node of the edge; speed report for that edge is generated. Vehicle waits for T_{edge} milliseconds and then vehicle jumps to the second node of the edge.

If the GUI is enabled, vehicles movement on the edge is shown on screen. To simulate this movement, the edge is divided into very small equal pieces by using equally distant hop points. Vehicle starts from the first node of the edge, and then in a loop, vehicle's position is jumped to the next hop of until vehicle moves to the end point of the edge. On each jump, vehicles position is refreshed in GUI. The jump interval and the number of hops are estimated in a proper way to achieve the determined driving speed.

In order to provide a smooth view and fluent tracking of the vehicle on the map, vehicle performs 25 hop jumps in a second. To achieve this value, agent waits 40 milliseconds between each hop. On each hop, GUI is refreshed which provides

1000/40 = 25 frames per second on the GUI of the vehicle agent. For an edge having length “d” meters, the number of hops required to simulate driving at “V” meters per second is calculated as follows;

$$H = (d * 25) / V$$

For example, let’s assume that vehicle is supposed to pass through an edge having length 62.32 meters with speed 55.2 km/h.

$$V = 55.2 / 3.6 \text{ m/s}$$

$$V = 15.333333 \text{ m/s}$$

$$H = (62.32 * 25) / 15.33333$$

$$H \cong 101 \text{ (decimal digits are truncated)}$$

Distance between each hop is 62,32 / 101 = 0,617 meters. Vehicle waits 40 ms between each hop. So in 1 second 0,617 * 25 = 15,425 meters of distance is traveled by the vehicle. This means a speed of 55.53 km/h, where difference caused by the truncation. Truncation is done to obtain a integer because the movement simulation loop needs an integer to iterate.

To simulate a realistic vehicle movement by using this method, FPS and agent waiting time between hops should satisfy the following equation;

$$FPS * T_{wait} = 1000 \text{ ms}$$

If fps is increased, T_{wait} needs to be decreased to satisfy this equation.

3.6 Routing Modes

The multi-agent system implemented in this study is a non deterministic system because the movement speeds and directions of the vehicles are randomly determined in a non deterministic way. When the system is launched in exactly same configurations, vehicles can move to different directions with different speeds on different times. Although the nodes, edges, distance between edges and the expected flow speeds of edges are fully observable by all vehicles, actual flow speeds of roads are changing dynamically. Vehicles can gather actual flow speeds by sending requests for a partial area. That’s why the flow environment is not fully observable.

There are three different routing modes implemented using A* search algorithm. A* search uses a distance plus heuristic cost function `f(x)` which is sum of two functions `g(x)` and `h(x)` where [38];

`g(x)` is the path cost function, cost from starting node to current node

`h(x)` is an admissible heuristic of the distance from current node to goal

Unlikely from Djkstra's shortest path algorithm [39], A* search algorithm does not explore the all nodes in the search three. Node exploration order is determined by h(x) function and node exploration stops when a path from start node to goal node is found. Vehicles are requesting the real-time flow speeds of edges in a circular area including start node and goal node. Nodes that are explored during A* search stays within this circular area. Briefly, A* search only explores the observable part of the map.

Each routing mode uses the same A* search algorithm but the cost and heuristic estimation between the nodes of the map is different in each mode. These modes are shortest path mode, fastest path mode and real-time fastest path mode.

Shortest Path Mode calculates a path by using the standard A* search in which both `g(x)` and `h(x)` estimation is done by considering distance. The path found in this mode is the shortest path.

Fastest Path Mode calculates a possibly faster path when compared to the shortest one. Route calculation is done by modifying the h(x) function of A* search in which `g(x)` cost is the duration required to come to current node from the start, `h(x)` is the duration required to fly from current node to goal. For a node x;

```
g(x) = g(y) + distance(y,x) / EFS(y,x)
h(x) = distance(x,goal) / MaxDrivingSpeed
```

where y is the previous node before reaching to x, EFS is the expected flow speed of the edge which connects nodes y and x. A heuristic is admissible when it has no chance to overestimate the cost to goal. In order to obtain an admissible h(x) in this mode, duration is calculated by dividing the air distance to maximum driving speed. 200 km/h is large enough value for MaxDrivingSpeed assuming that normal drivers

do not drive that much faster. The path found in this mode is the fastest path according to the expected flow speed of the roads in the map.

Real-Time Fastest Path mode is almost the same with the fastest path mode. The only difference is the `g(x)` function.

```
g(x) = g(y) + distance(y,x) / AFS(y,x)
```

where AFS is the actual flow speed of the edge between nodes y and x. This value is received from the server by Road Status Request Behavior. The path found in this mode is the fastest path according to the actual flow speed of the roads in the map.

One of these routing modes is selected when launching the agent. Active routing mode can be changed from the GUI, while the vehicle is moving.

3.7 Server

Server agent has an architecture which is similar to vehicle agents. Same road map is parsed and loaded into memory at the bottom level of the agent. This map is drawn on GUI by a similar map engine. In addition, there's a class named "*EdgeStatus*" to hold the report history and to calculate the latest average speed of an edge. For every edge object in the map, an instance of this class is generated. There exist two different behaviors which receive process and reply the messages coming from vehicle agents.

Difference between the vehicle agents' map engine and the server agent's map engine is that, a speed sign is drawn on each edge showing the average speed. Another difference is; color of the edges changes according the average speed. This feature allows the real time traffic density to be monitored visually on the map. A sample snapshot from the server agent GUI is given in Figure 13. In this monitoring system, roads flowing in their expected flow speed are painted in yellow. When traffic flows slower than it is expected, road color turns into red tones. Oppositely when traffic flows faster than it is expected, road color turns into green tones. Exact flow speed can be seen from the red speed signs on the roads.

In JADE, it is necessary to launch a main container in order to create a new multi-agent platform. In this multi agent system, the main container is created on the computer on which the server agent runs because server agent is the first agent that

needs to be created in the system. After the main container is launched and server agent becomes online, vehicle agents can join the platform.

The first action done by server agent is introducing itself to the yellow pages service which is provided by the DF agent. Agents provide a service registers themselves to DF agent by a service description. Other agents can query the DF agent in order to obtain the AIDs of agents which provide the service that they need to benefit. In this scenario, server agent register to DF by service name “RoadStatus”. Vehicle agents obtain the AID of server agent by querying the DF for agents providing service with this name. After server agent registers its service to DF, it starts to listen to the messages coming vehicle agents. Listening operation is done by two different behaviors.

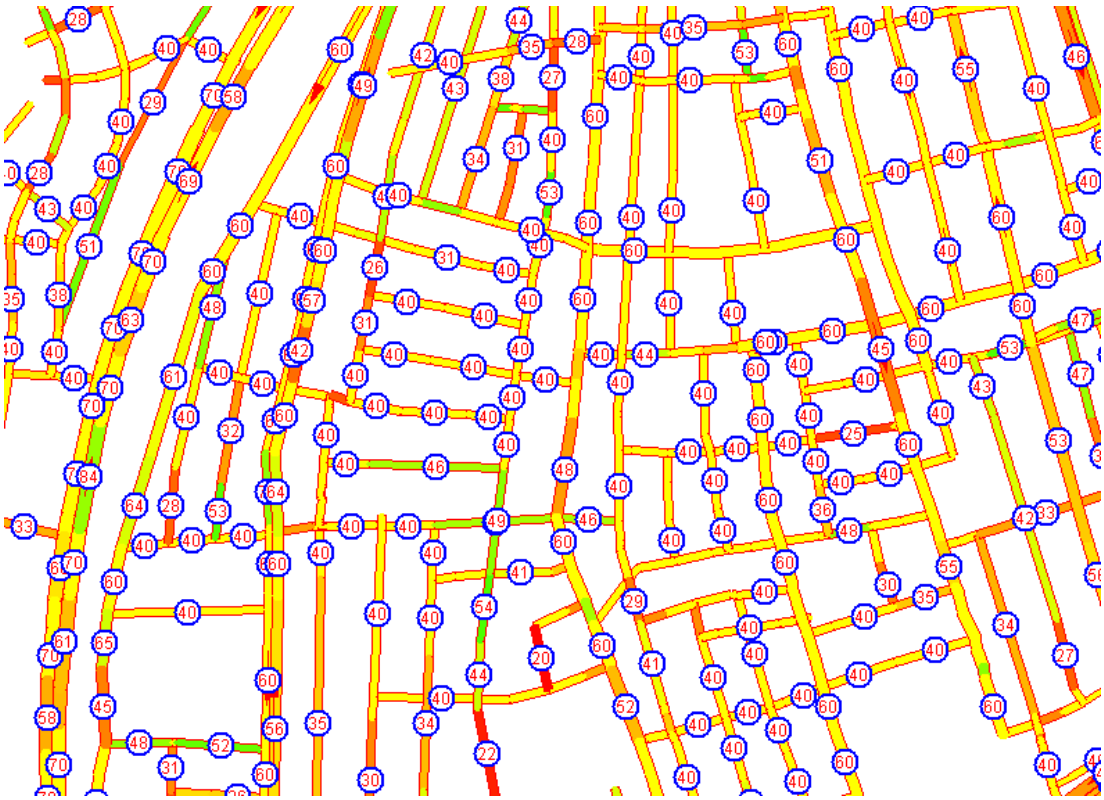


Figure 13 Screenshot of traffic density monitoring from server agent GUI

It is already explained four behaviors of vehicle agents. One of them sends speed reports to server agent periodically, one other sends request for latest road status. On server agent side, there are two behaviors to listen and process these two types of messages coming from vehicle agents.

Inform Message Handler Behavior is a cyclic behavior and responsible to retrieve, process the speed reports coming from the vehicle agents. In the action method of this behavior, message inbox of the agent is continuously checked for a new ACLMessage having "INFORM" performative. When a new message is available, it is polled from the queue and processed. Firstly speed report object is extracted from the message. This object contains the edge id, vehicle id, driving speed and the time that this report is generated. Secondly, the edge having this edge id is found, and then the "EdgeStatus" object inside this edge is accessed. "EdgeStatus" object has an internal queue to keep the history of the speed reports. When a new speed report for an edge is arrived, it is inserted into queue. If the queue is full, the oldest entry, which is in the head of queue, is removed from the queue, and then the new report is inserted. Common expire duration is defined for all "EdgeStatus" objects. A speed report object is deleted from the queue when expire duration is passed after the generation time of this report. If a speed report for the same edge from the same vehicle arrives, the old one is detected and updated instead of inserting duplicate entries. By these methods, only the latest and most up to date reports are stored in the queue. Figure 14 depicts the flow of steps in the process of message handling.

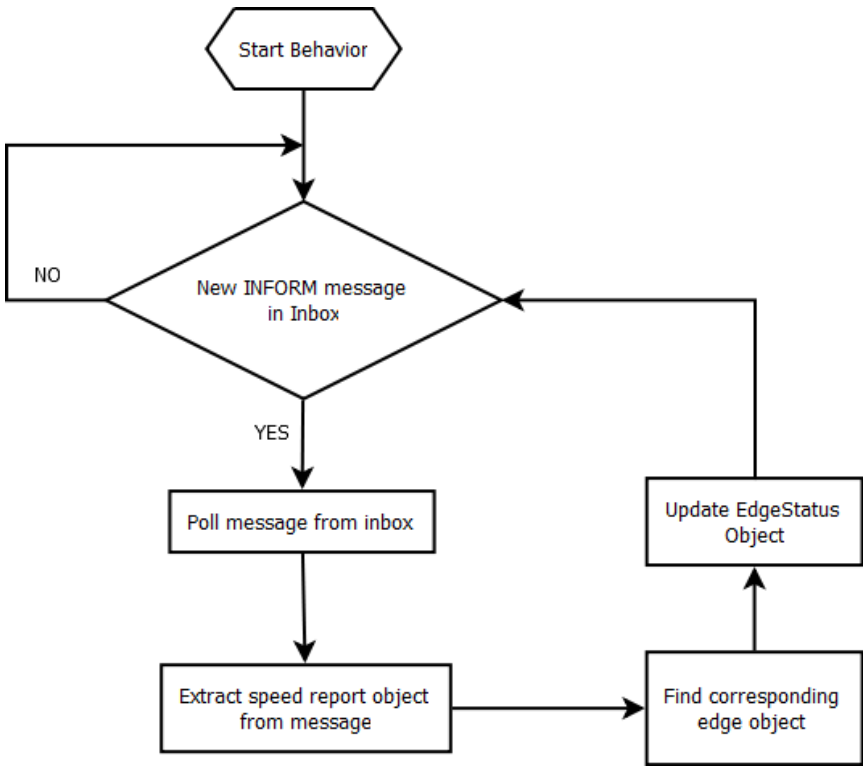


Figure 14 Flow Chart of Inform Message Handler Behavior

Request Message Handler Behavior is the second cyclic behavior of server agent which is responsible from listening, processing and replying the road status request messages coming from vehicle agents. In the action method of this behavior, message inbox of the agent is continuously checked for a new ACLMessage having "REQUEST" performative. When a new message is available, it is polled from the queue and processed. Firstly the array list object containing the edge ids is extracted from the message. Secondly, for each edge id in this list, the edge having this id if found and "EdgeStatus" object inside this edge is accessed. "EdgeStatus" class has a method which scans and refreshes the internal queue, calculates the average speed of the edge and returns it. For each edge in the list, corresponding average speed value is obtained by this method and written into a new array list. Finally this new array list is wrapped in a new ACLMessage having "INFORM" performative and sent to the sender of the request message. The behavior is depicted in Figure 15.

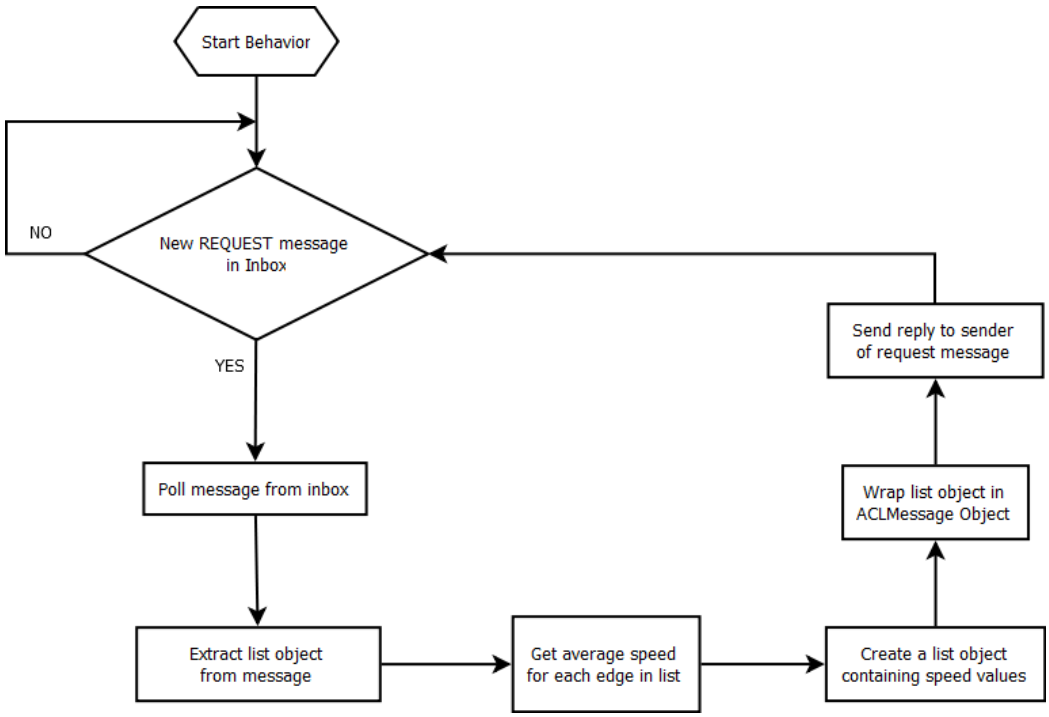


Figure 15 Flow Chart of Request Message Handler Behavior

3.8 Graphical User Interface

Vehicle agent has an optional GUI window. This GUI can be disabled or enabled when launching the agent. An activated GUI consumes significant amount of process power, so when launching many agents on the same computer, it is recommended to disable the GUI. A screen shot of the vehicle GUI is given in Figure 16.

In the middle of GUI window, there's a map panel on which the road map is drawn and the vehicles position is tracked when vehicle moves. If vehicle has an active route plan, it is also drawn by using a different color. Zoom level of the panel can be adjusted by mouse wheel. In the highest zoom level, map scale is 4:1 meaning that 4 pixels on the screen correspond to 1 meter distance. Yet, there is no limitation of zoom out. Shown area of the map on the panel can be shifted by mouse clicks. Center of the panel is accepted as reference point. When mouse clicked x pixels left and y pixels below the center, map is shifted to x pixels left and y pixels down.

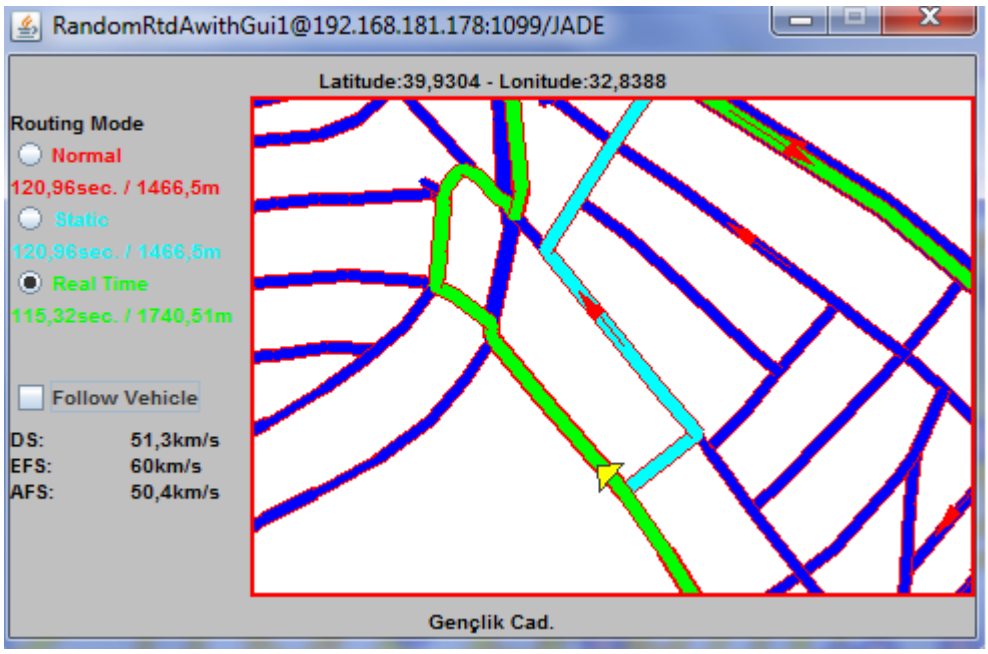


Figure 16 Sample Vehicle Monitoring Screen

On the top of the GUI window, there's a bar showing the coordinates of vehicle's location. On the bottom, there's another bar showing the name of the road on vehicle the vehicle is moving on. Finally on the left side of window, there's a small panel showing the route info and speed info of the edge that vehicle is moving on. In the routing mode part, there are three routing options which are mentioned in previous

section. Active routing method can be changed from here. Below each routing mode, estimated arrival time and route length is shown. Arrival time for all routing modes is estimated by using the AFS of the edges in the route.

In speed info part, there are three values shown.

- **DS** is the current driving speed of the vehicle.
- **EFS** is the expected flow speed of the current edge.
- **AFS** is the actual flow speed of the current edge. Shown value is taken from the latest road status reply received from the server.

In addition, there's a checkbox named "Follow Vehicle" on the left side of the window. When this checkbox is clicked, follow vehicle mode is activated. In this mode, vehicle moves, map is shifted in order to keep the vehicle on the center.

3.9 Logging Mechanism

Routed vehicle agents have a logging mechanism which provides information about their travel along that route. Logging is activated when start and destination node is given when launching the agent. Logs of each agent are written into a separate text file and contain launch timestamp of agent, arrival time to destination from the start, the length of the route and the average speed of the travel. This log file is stored on the computer on which the agent is running. Name of the log file is the local name of the agent. In the test phase, many runs are taken and result of each run is written into the same file by appending the logs of previous runs.

These log files are analyzed in Chapter 4 to measure the performance of the agents and different routing modes. Logging mechanism is not activated in randomly strolling agents because their performance is not measurable since they do not have a route plan.

CHAPTER 4

TEST RESULTS AND FINDINGS

In order to analyze the efficiency of the route planning based on real-time traffic data, several tests are conducted on the simulation developed. In this section, the testing environments, limitations, test cases, results and major findings are given.

4.1 Testing Environment

During the tests three distinct computers are used. Server executed on a powerful desktop computer having the following configurations:

- Intel Core2Duo E8400 3.0 Ghz processor,
- 4 gigabytes of RAM, and
- Windows XP Professional 32 bit operating system.

Random strolling vehicle agents which are collecting and reporting traffic data are executed on a laptop computer with

- AMD Turion TL-64 X2 2.1 Ghz processor,
- 2 gigabytes of RAM, and
- Windows 7 64 bit operating system.

Lastly, test agents are executed on a laptop computer having the following configurations:

- Intel Core i5 520M 2.4 Ghz processor,
- 4 gigabytes of RAM, and
- Windows 7 32 bit operating system.

In all machines, Java Runtime Environment Version 6 Update 20 is used.

4.3 Limitations

In the real-life scenario, each agent is assumed to run its own device. In this simulation, many agents are executed on the same computer during the tests. Because of this, there appeared some limitations in the number of agents in the platform and size of map used.

In JADE, each behavior runs a separate JAVA thread. The agent itself is another thread so that each vehicle agent creates multiple threads in the Java Virtual Machine (JVM). As the number of active threads in JVM increases, it is more likely to face thread synchronization problems and deadlocks appear in the runtime. That's why running more than 50 complex agents on the same container sometimes causes some threads of JADE to get blocked. When this happens, the message traffic between the agents stops. During the tests, life cycles of the threads running in Java Virtual Machine are monitored and any block state is detected. When a block state is detected, test run is restarted.

4.2 Map

In the scope of the study, apart from the simulation of real-time based route planning, a generic map drawing engine has been developed. This engine has been validated by drawing the real map of Ankara as explained in 3.2.2.

During the tests, an imaginary road map is generated as shown in Figure 17. Topology of the generated map has simpler and regular topology which is more likely a grid structure. There are 100 nodes in total, 180 edges between these nodes.

Road types in this map are primary, secondary and residential. The reason of this limitation is to keep the flow speed of roads closer, which makes the appearance of alternate routes more likely. For example, if a motorway passes from the middle of this map, most of the time fastest and real-time routes would use this very high speed road. Such a situation may prevent the appearance of alternate routes.

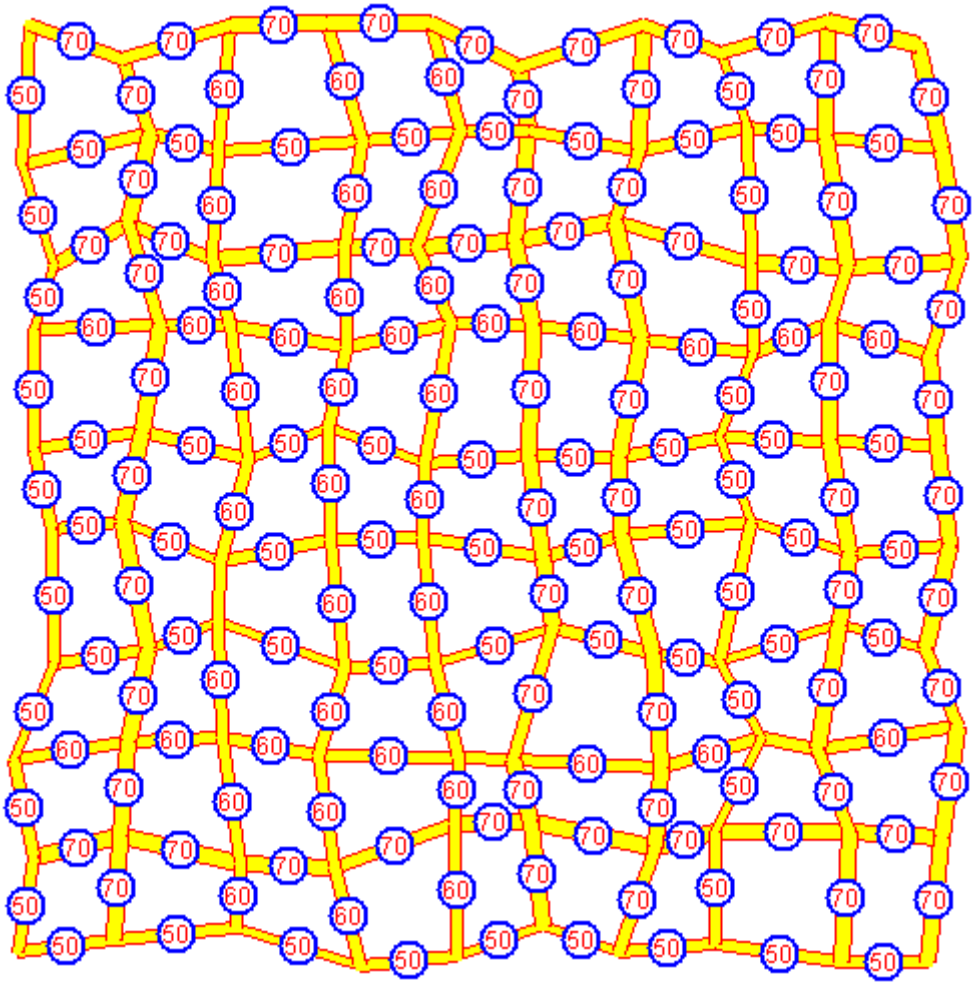


Figure 17 A Map for Testing

4.3 Test Cases

There are three fixed start nodes `s1`, `s2`, `s3` and three fixed destination d1, d2, d3 nodes are selected. Then three goals `g1`, `g2`, `g3` are defined by using these nodes. These goals are

```
g1 = {s1 -> d1}
g2 = {s2 -> d2}
g3 = {s3 -> d3}
```

There are nine test agents `TA1s`, `TA1f`, `TA1r`, `TA2s`, `TA2f`, `TA2r`, `TA3s`, `TA3f` and `TA3r` are defined. The goal assignment of these agents is shown in Table 2.

Table 2 Goal assignment of test agents

Agent Name	Goal	Route Mode
TA1s	g1	Shortest Path
TA1f	g1	Fastest Path
TA1r	g1	Real-Time Path
TA2s	g2	Shortest Path
TA2f	g2	Fastest Path
TA2r	g2	Real-Time Path
TA3s	g3	Shortest Path
TA3f	g3	Fastest Path
TA3r	g3	Real-Time Path

There are three test cases and each has different configurations. In all these configurations, test agents which are defined above and the imaginary test map are used. In other words, in each configuration, for each routing mode, three agents with different goals are executed. The difference between the configurations is the number of random strolling agents (called as x). Table 3 summarizes the configurations in each test case.

Table 3 Test Cases

Test Case #	# of random strolling agents (x)	# of nodes	# of runs
1	17	100	7
2	34	100	10
3	84	100	6

The basic process of each test case is as follows. Firstly the multi-agent platform has launched and server agent has joined to the system. Then, x random strolling agents are joined to the platform from another computer. After these random strolling agents strolls approximately two or three minutes in traffic and reports useful traffic information to server, test agents are joined to the system. Each agent firstly requests current traffic status from the server as it is explained in Section 3.4 and then starts moving from its start node to its goal node. During this movement, each agent

follows the route which is calculated according to routing modes assigned to it. When they arrive to their goals, they calculate the duration and length of their trip and write them into a log file. After all test agents arrive to their goals, these test agents are restarted with same goals without restarting the server and random strolling agents. Briefly, several test agent runs are taken in the same continuous traffic environment. Since the flow density of traffic varies during the time line, real-time route calculated by test agents may vary in each run. Unlikely shortest path and fastest path algorithms always gave the same routes in each run because they always use the same constant data.

4.4 Test Results

In each test case, several runs are taken and performances of three routing modes are compared. A detailed table showing the performance of each test result can be found at http://sites.google.com/site/ilterocal/testresults.rar and in the compact disc.

Results of agents having the same goal in test case 1 are compared in Table 4 and Table 5.

Table 4 Comparison of fastest vs. real-time routing for test case 1

	TA1f sec. - TA1r sec.	TA2f sec. - TA2r sec.	TA3f sec.- TA3r sec.
Run 1	0*	9,09	0
Run 2	0	-5,58	0
Run 3	0	0	0
Run 4	0	-1,04	13,47
Run 5	-0,26	2,91	0
Run 6	0	2,28	-1,59
Run 7	0	16,19	0
Average	**-0,26**	**3,98**	**5,93**

(*0 means that agents travelled along same route)

Agents TA1f and TA1r have given the same route for six times. Just once, they suggested different routes. In this run, TA1f arrived 0,26 seconds before TA1r.

Agents TA2f and TA2r have given different routes for six times. Just once, they suggested the same route. When different routes are suggested, TA2r arrived earlier for four times, TA2f arrived earlier for 2 times. In average, TA2r arrived 3.98 seconds earlier per run.

Agents TA3f and TA3r have given different routes for two times. In one of them, TA3r arrived 13.47 seconds earlier; in the other one TA3f arrived 1.59 seconds earlier.

Table 5 Comparison of shortest vs. real-time routing for test case 1

	TA1s sec -. TA1r sec.	TA2s sec. -. TA2r sec.	TA3s sec. - TA3r sec.
Run 1	57,28	52,28	32,15
Run 2	39,06	57,82	55,37
Run 3	155,79	65,47	36,96
Run 4	119,50	116,90	51,88
Run 5	39,19	80,74	49,62
Run 6	33,31	37,49	30,49
Run 7	44,39	111,63	29,61
Average	**69,79**	**74,62**	**40,87**

When arrival time of shortest path following agents and real-time route following agents are compared, it is very clear to see that real-time path provides a great benefit. Average benefit per run is 69.79 seconds for goal 1, 74.62 seconds for goal2 and 40.87 seconds for goal 3.

Table 6 shows the statistical comparison of fastest path and real-time path results.

Table 6 Statistical results in test case 1

Total Real Time Route Calculations	21
Number Different Fastest vs. Real Time Routes	9
Different Route Calculation Ratio	42,86%
Number of Real-Time Route Success	5
Real-Time Route Success Ratio	55,56%
Total Time Benefit by Real-Time Route	49,94 sec.
Total Time Loss by Real-Time Route	-8,48 sec.
Overall Benefit of Real-Time Route	41,46 sec.
Overal Benefit per a Real-Time Route	4,60sec.

The results of agents having the same goal in test case 2 are compared in Table 7 and Table 8.

Table 7 Comparison of fastest vs. real-time routing for test case 2

	TA1f sec. -. TA1r sec.	TA2f sec. -. TA2r sec.	TA3f sec. - TA3r sec.
Run 1	0	0	4,47
Run 2	0	-4,22	0
Run 3	-4,87	20,09	0
Run 4	3,48	36,58	0
Run 5	22,45	0	-3,65
Run 6	13,70	6,13	0
Run 7	17,67	18,34	0
Run 8	-0,67	-18,40	0
Run 9	0	0	2,65
Run 10	-2,11	0	0
Average	**7,09**	**9,75**	**1,16**

Agents TA1f and TA1r have given different routes for seven times. In four of them, TA1r arrived earlier; in remaining three TA1f arrived earlier. In average, TA1r arrived 7,09 seconds earlier per run.

Agents TA2f and TA2r have given different routes for six times. In four of them, TA2r arrived earlier; in remaining three TA2f arrived earlier. In average, TA2r arrived 9,75 seconds earlier per run. In run 4, real-time routing provided 36.58 seconds benefit.

Agents TA3f and TA3r have given different routes for three times. In two of them, TA3r arrived 4,47 and 2,65 seconds earlier respectively; in the other one TA3f arrived 3,65 seconds earlier.

When arrival time of shortest path following agents and real-time route following agents are compared, again it is very clear to see that real-time path provides a great benefit. Average benefit per run is 52,87 seconds for goal 1, 81,21 seconds for goal2 and 6,89 seconds for goal 3.

Table 8 Comparison of shortest vs. real-time routing for test case 2

	TA1s sec. - TA1r sec.	TA2s sec. - TA2r sec.	TA3s sec. - TA3r sec.
Run 1	83,16	83,10	47,14
Run 2	70,25	171,15	78,06
Run 3	37,44	59,94	35,55
Run 4	18,52	75,40	55,69
Run 5	65,24	44,52	29,48
Run 6	33,27	44,64	23,85
Run 7	61,81	88,22	58,17
Run 8	29,72	50,26	45,12
Run 9	85,16	65,24	53,17
Run 10	44,14	129,62	42,72
Average	**52,87**	**81,21**	**46,89**

Table 9 shows the statistical comparison of fastest path and real-time path results.

Table 9 Statistical comparison for test case 2

Total Real Time Route Calculations	30
Number Different Fastest Vs Real Time Routes	15
Different Route Calculation Ratio	50,00%
Number of Real-Time Route Success	10
Real-Time Route Success Ratio	66,67%
Total Time Benefit by Real-Time Route	145,62 sec.
Total Time Loss by Real-Time Route	-33,947 sec.
Overall Benefit of Real-Time Route	111,67 sec.
Overal Benefit per a Real-Time Route	7,446 sec.

The results of agents having the same goal in test case 3 are compared in Table 10 and Table 11.

Agents TA1f and TA1r have given different routes for three times. In all of them, TA1r arrived earlier; Real-time route brought benefit from 47,28 seconds up to 257,69 seconds. In average, TA1r arrived 139,71 seconds earlier per run.

Table 10 Comparison of fastest vs. real-time routing for test case 3

	TA1f sec. - TA1r sec.	TA2f sec. - TA2r sec.	TA3f sec. -. TA3r sec.
Run 1	0	0	1,96
Run 2	0	0	1,05
Run 3	0	0	8,08
Run 4	47,28	27,46	-7,23
Run 5	114,15	0,93	0
Run 6	257,69	0	5,42
Average	**139,71**	**14,19**	**1,85**

Agents TA2f and TA2r have given different routes for two times. In both of them, TA2r arrived earlier with a benefit of 27,46 and 0,93 seconds respectively.

Agents TA3f and TA3r have given different routes for five times. In four of them, TA3r arrived earlier respectively. Just for once TA3f arrived 7,23 seconds earlier. In average, TA3r arrived 1,85 seconds earlier per run.

Table 11 Comparison of shortest vs. real-time routing for test case 3

	TA1s sec. - TA1r sec.	TA2s sec. - TA2r sec.	TA3s sec. -. TA3r sec.
Run 1	51	109,61	88,94
Run 2	59,15	216,04	166,33
Run 3	175,60	343,14	243,63
Run 4	406,27	331,14	320,58
Run 5	339,29	301,53	144,24
Run 6	516,18	296,03	190,45
Average	**257,91**	**266,25**	**192,36**

When arrival time of shortest path following agents and real-time route following agents are compared, there appears a huge difference between real-time arrival times and shortest path arrival times that real-time. Average benefit per run is 257,91 seconds for goal 1, 266,25 seconds for goal2 and 192,36 seconds for goal 3.

Table 12 shows the statistical comparison of fastest path and real-time path results.

Table 12 Statistical comparison for test case 3

TEST RESULT SUMMARY	
Total Real Time Route Calculations	18
Number Different Fastest Vs Real Time Routes	10
Different Route Calculation Ratio	55,56%
Number of Real-Time Route Success	9
Real-Time Route Success Ratio	90,00%
Total Time Benefit by Real-Time Route	464,06 sec.
Total Time Loss by Real-Time Route	-7,230 sec.
Overall Benefit of Real-Time Route	456,83 sec.
Overal Benefit per a Real-Time Route	45,68 sec.

4.5 Findings

When the result summaries of these three configurations are compared, there are several findings observed depending on the number of random strolling agents. Result of fastest path and real-time routing in each configuration is evaluated separately and then a general observation is done by considering all of them.

In the first set-up, nine alternate routes have been calculated by real-time routing and five of them performed better than fastest routing mode. Success ratio of real-time routing against fastest routing is 55,56% in this configuration. Overall time benefit per run is calculated as 4.60 seconds.

In second configuration, 15 alternate routes have been calculated by real-time routing and 10 of them performed better than fastest routing mode. Success ratio of real-time routing against fastest routing is 66,67% in this configuration. Overall time benefit per run is increased to as 7,44 seconds.

Lastly in third configuration, 10 alternate routes have calculated by real-time routing and nine of them performed better than fastest routing mode. Success ratio of real-time routing against fastest routing is 90% in this configuration. Overall time benefit per run is exploded to 45,68 seconds because of the individual results of agents having goal 1. Although number of agents has an effect on this sharp increase in the arrival time gap between fastest and real-time, the over-riding reason of this gap is the randomness in the speed determination of vehicles. In this configuration, it is seen that route difference between fastest and real-time mode occurs starting from

the 4th run. Until the end of 3rd run, traffic should have been flown in a speed close to expected flow speed. From that point, flow speed of the road pieces on the route suggested by fastest routing mode decreased a lot because vehicles passing from those roads should been determined slower movement speeds. This incident caused TA3f to move slower on its path but TA3r avoided this incident by using real-time speed flow data.

It is observed that when the number of traffic data collecting agents in the system increases, the success ratio of real-time routing against fastest routing increases too. It is also observed that time benefit of real-time routing is directly proportional to the number of data collecting agents. When the number of agents in the traffic increases, more speed reports arrive to server agent. This enables to server to update average flow speed of roads more frequently which provides fresh data to the vehicle agents for their routing. In all configurations, both fastest and real-time routing modes performed far better than shortest routing.

To sum up, this simulation results show that shortest path is not useful when arrival time to destination is the evaluation criteria. Considering road flow speeds in route calculation enables vehicles to arrive their destination earlier. In addition, considering real-time flow speed of the roads provides a noticeable improvement on the arrival time of vehicles. Finally, it is seen that efficiency of the real-time traffic data collection and usage varies by the number of data collecting agents.

CHAPTER 5

DISCUSSION AND CONCLUSIONS

5.1 Discussion

According to the findings, it is figured out that route planning based on real-time traffic data yields more effective results when enough data is collected. Vehicles moving according to real-time route planning are mostly arriving to their destinations earlier. Test results show that increase in the amount of data collected has a positive effect on the efficiency of the route planning using real-time data. Another factor that affects the efficiency of real-time routing is the flow dynamics of the traffic. When traffic flow is in expected speed ranges, benefits of real-time routing is not much significant. However, when traffic flow speed is unexpectedly slow in some regions of roads; benefits of using real-time routing starts to become more noticeable.

In real life, traffic flow slows down because of many reasons such as accidents, road constructions and maintenance, traffic jams, irregularly parked vehicles and so on. Drivers who are spending a lot of time in city traffic can predict the possible traffic jams at specific regions of city on specific times and they may try to avoid such jams by using alternate routes to arrive their destination earlier. However, the other incidents are not predictable by drivers and drivers are not aware of them unless they hear from an external source of information.

In the simulation done, traffic has a chance to slow down in some part of roads as it happened in the test case 3. This chance is created by using a non-deterministic movement simulation algorithm. Vehicles agents are querying the speed flow of

roads around them from server and they are able to avoid slow downed roads if faster route to their destination exists.

In this study, a multi-agent architecture has been designed and proposed to utilize real-time data collection and dissemination approach in real word mobile devices. According to the design proposed, the multi-agent system collects processes and distributes traffic flow speed information in a centralized way. The proposed architecture enables the agents to establish communication with central agent in order to request and propagate information about traffic. Overview of this architecture is shown in Figure 18.

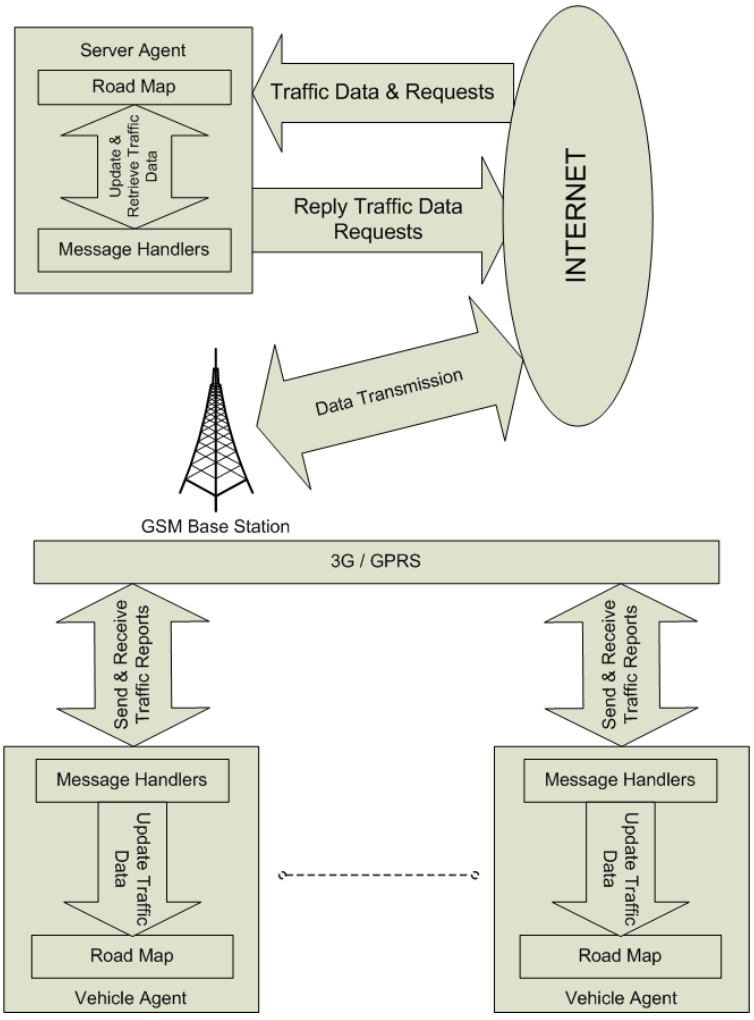


Figure 18 Overview of Multi-Agent architecture.

There are two types of agents in this architecture which are vehicle agents and server agent. Vehicle agents and server agent are designed to communicate with each other

over internet by using TCP/IP protocol. Every device that has vehicle agent running on it must have an internet connection. Roles of these agents in this architecture are briefly explained below.

Vehicle agents: In this scenario, vehicles or drivers having a navigation device which has internet access with 3G will act as an agent. Vehicle agents are assumed to run on portable wireless devices that are capable of connecting to internet and receiving GPS signals such as PDAs, smart phones, portable navigation devices. Their roles are to receive GPS signals that broadcasted from satellites, to determine the coordinates of the vehicle on the earth, to show the position of the vehicle on a map, to calculate and show route on demand, to show details about the active route plan. In the background, those agents will report the traffic flow speed of the road segment at which they drive along, to a centralized server and will also request and receive the traffic flow speed of specific set of road segment from the server.

Server agent: The architecture is based on this agent who is assumed to run on a powerful centralized server with a reliable network connection. The role of this agent is to receive traffic reports from the vehicle agents, to keep average speed data for all road segments in the map, and to reply the traffic data requests coming from the vehicle agents. Every vehicle agent will communicate with this agent during their travel in the traffic.

The simulation software developed in this study is a prototype for the architecture above. The simulator can be used and improved by other researchers who have intermediate level of java programming skills and JADE knowledge. Simulation software is not designed as a general purpose simulator that can be fully controllable from user interface. Some basic parameters can be changed from configuration files. Researchers can implement new algorithms and test them by using graphical vehicle tracking interface of the simulator. Other modifications in the running mechanism of the simulator needs code changes. A tutorial explaining how to setup, configure and run the simulator is given in Appendix.

5.2 Conclusions

Road traffic is a complex and unpredictable environment in which road conditions can change depending on a variety of causes. Drivers moving in traffic wishes to arrive their destination as quickly as possible. In such a dynamically changing environment, travelling between two points in the least possible time is a challenging problem. Shortest path between two points is usually not the fastest path. Similarly, routing algorithms which uses statistical and historical data are not able to guarantee the fastest path at a given time. Therefore, usage of real-time traffic information when finding the fastest path between two points is a possible way of finding the fastest route.

In this study, a multi agent simulation is designed and developed in which real-time traffic data is collected and disseminated between agents. Traffic data that is collected and disseminated between the agents is used to calculate real-time fastest route between two points. The efficiency of the calculated real-time fastest route is tested by comparing its performance by shortest and statistical fastest routes. Test results show that vehicles using the route calculated by using real-time data are arriving to their destination earlier than the vehicles using other routing methods. It is observed that the time gained by using real-time data is dependent on the collected data and condition of the roads on possible routes. When more agents participate in data collection, collected data becomes more updated and this increases the quality of the route calculated. It is also observed that quality of real-time routing increase when flow speed some of roads on the possible route slows down. Briefly, it is possible to say that usage of real-time traffic information is beneficial in estimating the fastest route.

All traffic reports are submitted to server agent meaning that server agent is always keeping the latest flow speed average of the roads. As a future work, a logging mechanism is planned to implement which records the average flow speed of each road piece for small intervals in timeline. These logs would be analyzed to find a numerical relation between the quality of real-time route calculation and the condition of roads in possible routes.

Furthermore in this study, a generic XML based map format and a map drawing engine developed in JAVA. These two components can be used in projects and research studies which requires the usage of a real road map based on WGS84 coordinate system. The map format used stores a very limited type of information which satisfy the needs of this study. This format can be extended to store more type of map information in the future. Visual map drawing engine has some performance problems which need to be solved by using optimization techniques. If this map drawing is engine is adapted and used to a mobile application, those performance problems may prevent the application to run fast enough on a mobile device. Hence, this map engine needs to be improved and optimized to be used in mobile devices and applications.

Lastly, a navigation application for mobile phones is planned to be implemented in the future by using some components of this simulation software fully or partially. Some minor changes in the internal calculation functions and some major structural changes in the GUI component are required to convert this software to a JAVA ME application. Exact performance and benefit of real-time data usage in route planning can be observed by testing the multi-agent system in real-life.

REFERENCES

[1] Onstar General Motors Smart Car http://www.onstar.com, last retrieved 2009.

[2] Xiaolei Li, Jiawei Han, Jae-Gil Lee and Hector Gonzalez, Traffic Density-Based Discovery of Hot Routes in Road Networks,SSTD,2007,pp.441-459.

[3] Gonzalez, Hector, Han, Jiawei, Li, Xiaolei, Myslinska, Margaret and Sondag, John Paul, Adaptive fastest path computation on a road network: a traffic mining approach, VLDB '07: Proceedings of the 33rd international conference on Very large data bases, 2007, pp.794--805, VLDB Endowment.

[4] Kriegel, Hans-Peter; Renz, Matthias; Schubert, Matthias & Züfle, Andreas: Statistical Density Prediction in Traffic Networks. SIAM, 2008 , pp.692-703.

[5] Chia-Hao Lo, Wen-Chih Peng, Chien-Wen Chen, Ting-Yu Lin, Chun-Shuo Lin: CarWeb: A Traffic Data Collection Platform. MDM 2008, pp.221-222

[6] TomTom IQ Routes, last retrieved 2010 July http://www.tomtom.com/whytomtom/topic.php?topic=5&subject=3

[7] TomTom HDTraffic, last retrieved 2010 July http://www.tomtom.com/services/service.php?id=2

[8] TomTom HDTraffic While Paper, last retrieved 2010 July http://www.tomtom.com/lib/doc/download/HDT_White_Paper.pdf

[9] NAVTEQ Traffic, last retrieved 2010 July http://corporate.navteq.com/products_data_advanced_traffic.htm

[10] Msn Direct, last retrieved 2010 July http://www.msndirect.com/services.aspx

[11] XM NavTraffic, last retrieved 2010 July http://www.xmradio.com/navtraffic/

[12] Huaying Xu Barth, M.An adaptive dissemination mechanism for inter-vehicle communication-based decentralized traffic information systems. Intelligent Transportation Systems Conference, 2006. ITSC '06. IEEE pp.1207-1213

[13] von Toorenburg J, der Linden R., van Velzen B.Predictive control in traffic management. Final Report AV-2468,1996 Ministry of Transport, Public Works and Water

[14] Management, The Netherlands, 1996.Ahuja R., Magnanti T, and Odin J. Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[15] Chem S., Sun Z. and Bridge B. Automatic traffic monitoringby intelligent sound detection. In Proceedings of the 1997 IEEE Conference on Intelligent Transportation Systems.

[16] Forren J. and Jaarsma D. Traffic monitoring by tire noise. In Proceedings of the 1997 IEEE Conference on Intelligent Transportation Systems.

[17] Nishizawa S., Cheok K., Young W. and Zhao W. Traffic monitor using two multiple-beam laser radars. In Proceedings of the 1997 IEEE Conference on Intelligent Transportation Systems.

[18] Hancock T., Judd S., Novak C., Ricard S. Automatic vehicle location using cameras. In Proceedings of the 1997 IEEE Conference on Intelligent Transportation Systems.

[19] Moukas, A., Chandrinos, K. and Maes, P.Trafficopter: A Distributed Collection System for Traffic Information. CIA '98: Proceedings of the Second International Workshop on Cooperative Information Agents II, Learning, Mobility and Electronic Commerce for Information Discovery on the Internet, 1998, Springer-Verlag, pp. 33-43

[20] L. Fu, D. Sun, L.R. Rilett, Heuristic shortest path algorithms for transportation applications: State of the art, Computers & Operations Research Vol. 33, Issue 11, 2006, pp. 3324-3343

[21] Guzolek J, Koch E. Real-time route planning in road network. Proceedings of VINS, September 11–13, 1989. Toronto, Ontario, Canada, pp. 165–9.

[22] Hart, P.E. Nilsson, N.J. Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths, Systems Science and Cybernetics, IEEE Transactions on, Vol. 4 Issue 2, July 1968, pp 100-107

[23] Bigus, J.P. (2000). Agent Building and Learning Environment. Proceedings of the International Conference on Autonomous Agents 2000, Association for Computing Machinery, p108-109. Barcelona, Spain.[24] AgentBuilder (2000), last retrieved 2010 July http://www.agentbuilder.com/Documentation.

[25] Lange, D. B. and Mitsuru, O. 1998 Programming and Deploying Java Mobile Agents Aglets. 1st. Addison-Wesley Longman Publishing Co., Inc.

[26] Stefan Poslad, Phil Buckle, Rob Hadingham, The FIPA-OS agent platform: Open Source for Open Standards, 2000

[27] The Foundation for Intelligent Physical Agents FIPA, last retrieved 2010 July http://www.fipa.org/

[28] Java Agent Development Environment JADE, last retrieved 2010 July http://jade.tilab.com/

[29] JATLite: A Java Agent Infrastructure with Message Routing, Heecheol Jeon,Charles Petrie,Mark R. Cutkosky, IEEE Internet Computing, 4(2), 2000.

[30] FIPA Specifications, last retrieved 2010 July http://www.fipa.org/specs/fipa00037/SC00037J.html

[31] JADE Programmers Guide, page 7, last retrieved 2010 July http://jade.tilab.com/doc/programmersguide.pdf

[32] World Geodetic System 84, last retrieved 2010 July http://en.wikipedia.org/wiki/World_Geodetic_System

[33] Google Earth, last retrieved 2010 July http://earth.google.com

[34] Google Maps, last retrieved 2010 July http://maps.google.com

[35] OpenStreetMap, last retrieved 2010 July http://www.openstreetmap.org/

[36] Creative Commons Attribution-ShareAlike 2.0 license, last retrieved 2010 July http://creativecommons.org/licenses/by-sa/2.0/

[37] SAX XML Parser, last retrieved 2010 July http://www.saxproject.org/

[38] A* search algorithm, last retrieved 2010 July http://en.wikipedia.org/wiki/A*_search_algorithm

[39] Dijkstra's search algorithm, last retrieved 2010 July http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

APPENDIX

This appendix section explains how to deploy, run and modify the simulation software.

Compact Disc Contents

Compact disc provided with this thesis contains the following;

- Eclipse 3.5.1 IDE for Windows platform
- Eclipse project and source files of the simulator including JADE libraries
- Map files in xml format
- Detailed test results
- PDF copy of this thesis

Deploying the Simulator

Before deploying the simulator, be sure that Java SE Development Kit (JDK) is installed on your operating system. If it is not installed, download and install it from "http://www.oracle.com/technetwork/java/javase/index.html". Afterwards, open the file named "378085.rar" in the root folder of compact disc by using WinRAR program. Extract the folder named "eclipse" and "project" to any location on your hard drive. Launch Eclipse by double clicking on eclipse.exe in the extracted folder. When you run Eclipse for the first time, it asks you to choose a workspace for Eclipse. Create a workspace folder on your hard drive and choose this folder as workspace by setting it as default workspace. After choosing the workspace, welcome page of Eclipse appears. Close the welcome page and now you can see the package explorer of Eclipse on the left side. Open the "File" tab from the main menu

and select “Import”. From import window, go into “General” folder and select “Existing Projects into Workspace”. A new window opens which lets you to choose the root directory of the existing eclipse project. Browse and select the “project” folder which you have extracted at the beginning. Click the finish button at the bottom of import window. Now you can see the project named “Thesis” in the package explorer window.

Configuring the Vehicle Agents

There are two files to configure the vehicle agents’ parameters and their creation process. First one is a java class named “SimulationConstants”, the other one is “config.txt”.

In Eclipse, double click on “Thesis” project from the package explorer, and then double click the “src” folder. Under “src” folder, you can see the java packages. In “otherClasses” package, there’s a file named “SimulationConstants.java” in which the parameters of vehicle agents such as report and request periods, speed variance values, FPS, movement hop period are stored. Value of those parameters can be re-assigned from this file. Double click on the file to edit it. Editor window opens on the right side of the package explorer. When the file is saved, it is re-compiled and changes are applied on the next run.

There is another agent in this simulation named “VehicleAgentCreator” (VAC) which creates the vehicle agents in the simulation platform. In order to create many vehicle agents in the JADE platform, a VAC agent is needed. Under “src” folder you can see the “config.txt” file which stores the parameters about creating vehicle agents in the simulation. VAC agent parses “config.txt” file and creates vehicle agents according to the values written in this file. The amount of agent in the system, their types, time to join the system, start and goal nodes are written in this file. These parameters can be modified in this file. In this file we can specify the following agent creation parameters;

- Delay time T_{delay} between creating agents
- Waiting time T_{wait} between creating the last random strolling/routed agent and test agents

- Number of random strolling agents with GUI and without GUI. Start node of these agents are selected randomly.
- Number of randomly strolling routed agents with GUI and without GUI. Start node of these agents are selected randomly.
- Start node ids of random strolling agents having a start node. GUI's of those agents are disabled. If "n" ids specified, "n" agents of this type is created. If no ids specified, no agents of this type created.
- Start and Goal node id pairs of routed test agents. If "n" id pairs specified, "n" agents of this type is created. If no ids specified, no agents of this type created.

Start node and goal node ids given in this file must be must be valid existing nodes in the road map file.

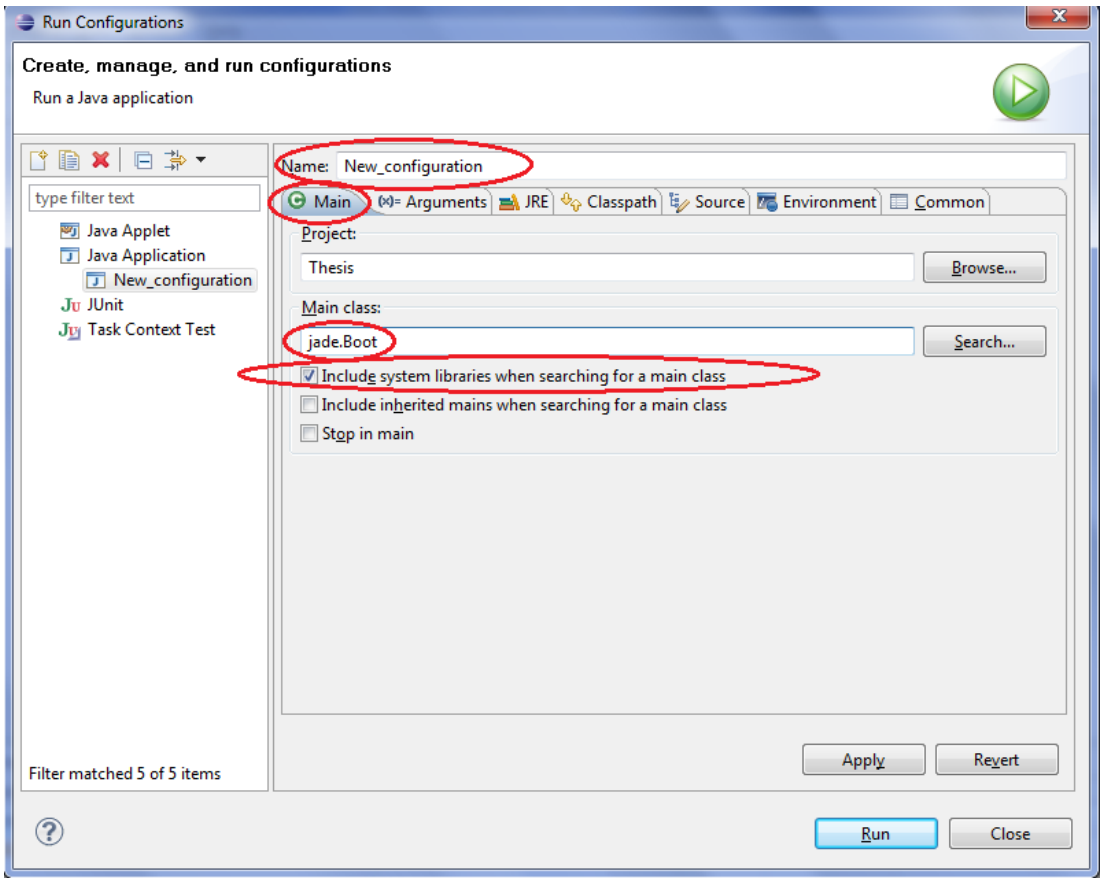
According to those parameters, VAC firstly creates the random strolling agents having random start nodes. Then it creates the routed agents having random start and goal nodes. After that, random strolling agent with given start nodes are created if they are specified. Up to this point, VAC waits T_{delay} milliseconds between the creations of each vehicle agent. Before starting to create the test agents, VAC waits T_{wait} milliseconds. Aim of this waiting is to allow the previously created agents to collect enough traffic data before the test agents become alive. Finally the test agents are created by parsing their start and goal nodes. More explanation about the parameters and the syntax of the file is written at the end of this file.

Running the simulation

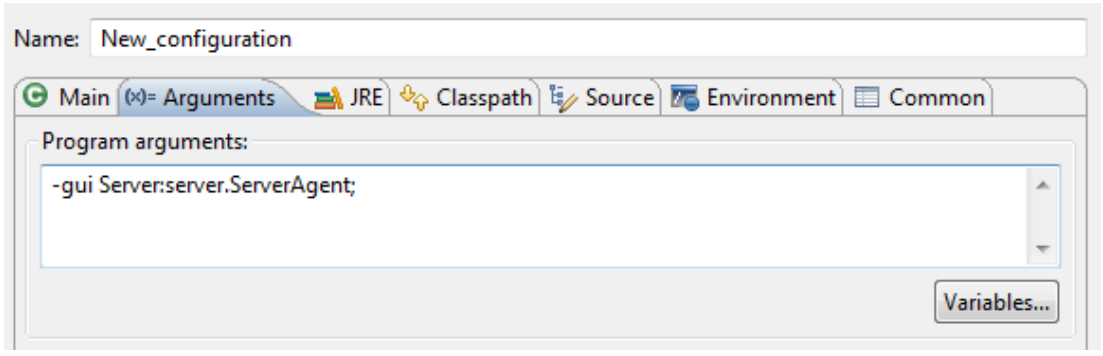
In this part, it is explained how to run the simulation with different server and vehicle agent combinations.

A container is launched by running the main class of JADE named "jade.Boot" and passing arguments to it. This operation can be easily done by using Eclipse. From the package explorer, right click to Thesis project and go onto "Run As" menu; and then click to "Run Configurations" in the submenu. In the window opened, double click to "Java Application" option from the left side. A new run configuration named "New_Configuration" is now created. You can change its name from the top of

window. Under the configuration name, you will see several tabs. In the “Main” tab, you must specify the project name, main class of the project as “jade.Boot” and click the include option as shown below.



In the arguments tab, we can enter the options for the container and we can also provide agent specifiers to create agents in the container. After setting the arguments, we can launch a new container by clicking the Run button at the bottom.



If a host IP is provided from the argument line, launched container joins to an existing platform whose main container is running on that host. If host IP is not

provided, a new JADE platform is launched and this container becomes the main container of the platform.

```
-gui -name SimulationPlatorm -container-name ServerContainer
Server:server.ServerAgent;
```

The example argument line above launches a new JADE platform with name “SimulationPlatorm” and creates the main container with name “ServerContainer”. Note that there’s no host IP is provided. The option –gui shows the debugging GUI of the JADE. The last phrase of this line is an agent specifier which creates the server agent in this main container. Syntax of agent specifier is as follows

```
<agent nickname>:<package name>.<agent class name>
```

Assuming that a platform and main container is launched by using the argument line above on a computer having IP “192.168.1.100”, we can create a new container on another computer to host vehicle agents as follows;

```
-container -host 192.168.1.100 –container-name
VehicleContainer1 VAC1:VehicleAgentCreator
```

The example argument line above creates a new peripheral container named “VehicleContainer1”, registers it to the main container hosted on “192.168.1.100” and then creates a vehicle agent creator agent named “VAC1”. VAC1 parses the “config.txt” file on the current machine and creates vehicle agents in this container. By this method, many vehicle agent creators can be created on different containers in order to distribute vehicle agents on different computers. Log files of test agents are created in a folder named “logs” which is placed under the “project” folder namely “<agent name>.txt”. Sample log file content is given below.

```
Run taken at time 1280406494274
Arrived to destination in 253.069seconds
Route Lenght: 4102.5603475691405 Average Speed:
58.36043628911051km/h
-----------------------------------------
```

In case of the availability of only one computer, both the server agent and vehicle agents on the same host by giving two agent specifiers as follows;

```
-gui -name SimulationPlatorm
Server:server.ServerAgent;VAC:VehicleAgentCreator
```

Using Alternate Map Files

There are different road maps are used during this study for different purposes as explained in previous sections. In “378085.rar” file, there’s a folder named “maps” which contains alternate maps for the simulator. Alternate maps are as follows;

- An alternate test map having 400 nodes and grid like topology
- Map of Ankara including whole city center inside the surrounding motorway. This map contains dead ends and duplicate nodes, which causes problems. It’s not recommended to use it as it is.
- Partial map of the region “Çankaya” of Ankara having same problems as above. It’s not recommended to use it as it is.
- Map of Bahçelievler district having the problems above are eliminated.

In the existing eclipse project the test map introduced in section 4.2 is used as default map. Map file is named “MapFile.xml” and located in the project folder under “src” folder. In order to use the alternate maps, rename the one you want as “MapFile.xml” and overwrite the one in “src” folder.