SOLVING VEHICLE ROUTING PROBLEM USING MULTI AGENT TECHNOLOGY

Amith Chanaka Mendis

129106V

Degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

University of Moratuwa
Sri Lanka

November 2016

SOLVING VEHICLE ROUTING PROBLEM USING MULTI AGENT TECHNOLOGY

Amith Chanaka Mendis

129106V

Thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Artificial Intelligence

Department of Computational Mathematics

University of Moratuwa
Sri Lanka

November 2016

Declaration

I declare that this interim report does not incorporate, without acknowledgment, any material previously submitted for a Degree or a Diploma in any University and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my interim report, if accepted, to be made available for photocopying and for interlibrary loans, and for the title and summary to be made available to outside organization.

Name of Student	Signature of Student
W. A. C. Mendis	Date:

Supervised by

Name of Supervisor(s)	Signature of Supervisor(s)
Prof. Asoka S. Karunananda	Date:

Acknowledgements

I will be failing my duty if I do not thank the people who have given me remarkable support and help to make my project a successful one. At the outset, I wish to thank MillenniumIT Software (Pvt) Ltd for giving me the idea to come out of project this nature. Also I wish to offer my humble gratitude to Prof. Asoka Karunananda for the guidance and encouragement given to me. At last but not least, I wish to thank all my friends who have shared their knowledge and helped me in several ways to complete this process.

Abstract

In today's world transportation plays an important role in logistics and it appears in various sections of logistics processes. It occupies one-third of the amount in the logistics costs and influence the performance of logistics system hugely. Therefore through better transportation planning businesses can improve their customer experience (service level) and reduce the overall logistic cost. Companies are using people to do this task manually. When number of destinations and number of transport vehicles are high, route planner has to find lots of information (information about roads, distance between destinations, traffic condition of roads, etc.…) and synthesize them manually to find the solution. Therefore this task is became very time consuming and produces inefficient solutions most of the time. Because of these factors, it requires lots of human intervention and wasting lot of time and money because of in-efficient route designs.

This research studies how multi agent technology can be used to overcome the above identified issue. Existing automated solutions for vehicle routing problem like Tabu search (TS), genetic algorithm (GA), and evolutionary algorithms (EA) uses destination point and vehicle details as data points, but with multi agent technology these data points convert to agents who can negotiate and take decisions collaboratively. Because it has features like autonomy, negotiation and emergent property, it introduces autonomy to the system and comes up with best or near best solutions as emergent properties through negotiations.

As the subject of this study 8.00 pm transport planning of MillenniumIT Software (Pvt) Ltd was chose and process of planning routes is automated using multi agent technology. 8.00 pm transport requests are different for each day. Therefore route plan should change day by day to cater the requirement. As a solution operation team of MillenniumIT generates manual route plans for each day and it is a challenging task because of the high number of passengers and vehicle are increasing the complexity of the problem. In automated system it used information about vehicles (number of vehicles, capacity of each vehicle) and passengers (passenger name, latitude of destination. longitude of destination) as inputs. After the requesting process is over system generates agents for each passenger and vehicle. Then those agents are developing a solution as an emergent property through negotiation and as output it generates route plans for 8.00 pm transportation of given day.

Then shuttle request data for 10 days were selected randomly as sample dataset to evaluate the automated solution. Then manually generated and automated shuttle plans for those shuttle requests were collected, calculated the total route distance and compared against each other.

The results show that 8 times out of 10 automated route plan is cost effective than the manually generated plan. Therefore it was concluded that Vehicle routing problem can solve by multi agent technology.

Table of Content

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Prolegomena

Vehicle Routing Problem (VRP) is an optimization problem and it generalizes the well-known Travelling Salesman Problem (TSP) [3]. VRP is about designing least cost routes from one depot to a set of geographically scattered points. The routes must be designed in such a way that each point is visited only once by exactly one vehicle within a given time interval and the total demands of all points on one particular route must not exceed the capacity of the vehicle. The objective of this problem is to minimize the number of vehicles required, minimize the total cost of all routes and to maximize the number of demands transported [15].

Dantzig and Ramser (1959) were the first to introduce the “Truck Dispatching Problem", modeling how a fleet of homogeneous trucks could serve the demand for oil of a number of gas stations from a central hub and with a minimum travelled distance [1]. Five years later, Clarke and Wright (1964) generalized this problem to a linear optimization problem that is commonly encountered in the domain of logistics and transport: i.e., how to serve a set of customers, geographically diffused around the central depot, using a fleet of trucks with varying capacities [2]. This became known as the “Vehicle Routing Problem” (VRP), one of the most widely studied topics in the field of Operations Research.

The current VRP models, however, are immensely different from the one introduced by Dantzig and Ramser (1959) and Clarke and Wright (1964), as they increasingly aim to incorporate real-life complexities, such as for instance, time-dependent travel times (reflecting to traffic congestion), time windows for pickup and delivery, and input information (e.g., demand information) that changes dynamically over time . These features bring along substantial complexity. As the VRP is an NP-hard problem [4], exact algorithms are only efficient for small problem instances. Heuristics and metaheuristics are often more suitable for practical applications, because real-life problems are considerably larger in scale (e.g., a company may need to supply thousands of customers from dozens of depots with numerous vehicles and subject to a variety of constraints).

The number of solution methods introduced in academic literature (for old as well as new variants of the VRP) has grown rapidly over the past decades. However, most of the solutions are specific for special scenario of Vehicle Routing Problem and they can be only used to solve those specific problems. Since there are no generalized and cost effective VRP solving software solutions in the market, most of the transporting companies are trying to solve this issue manually. However, the approach is very costly and time consuming. Due to the complex and dynamic nature of this problem humans and traditional computer programs are unable to give the most efficient solution.

This research proposes a solution to the above issue using Multi Agent Technology. The main objective of this research is to design routes for the Vehicle Routing Problem which is efficient while fulfilling individual needs of engaging parties.

1.2 Aim & Objectives

Aim **:** The aim of this project is to develop a Multi Agent based solution to solve the Vehicle Routing Problem (VRP) while considering the uniqueness of the needs of parties that are involved.

Objectives:

- Study the Vehicle Routing Problem and existing solutions.
- Study of technologies that can solve the problem (Multi Agent Systems).
- Design and develop a system which solves the problem.
- Evaluation of the proposed solution.
- Preparation of the final documentation.

1.3 Background and Motivation

In today's world transportation plays an important role in logistics and it appears in various sections of logistics processes [18]. It occupies one-third of the amount in the logistics costs and influence the performance of logistics system massively. Therefore, well planned transportation could reduce operation cost, and promote service quality. The Vehicle Routing Problem (VRP) is one of the NP-hard problems which can be seen throughout various areas of transport and operational domain. Globally most of companies and organizations are struggling with similar issues and they are trying to solve it manually. Nevertheless, these issues are dynamic, complex and they can have many solutions. Due to this nature of the issue, generating a solution manually is difficult and it consumes a lot of time and effort. As a result, it can have multiple possible answers, finding the best solution can be challenging. Additionally, at times the solution which is chosen as the best solution will not be the best solution at all.

MillenniumIT Software (Pvt) Ltd is one of the leading software development Companies in Sri Lanka which develops and maintains softwares for international capital markets (i.e.: Colombo Stock Market, London Stock Exchange Group). It provides transportation service at 8.00 pm for employees who work after 6.00 pm. additionally; employees can request for transport through an internal system before 7.45 pm. Till 7.45 pm an employee can change his or her request or cancel the request. After 7.45 pm employees cannot change their request and the operational team starts to create transportation plans manually for 8.00 pm transportation. At 8.00 pm, the transportation plan is shared across drivers. Due to the higher number of requests and limited time, it is very challenging to manually generate a cost effective and employee satisfying vehicle routing plan.

Therefore, most of the time manually generated solutions are not cost effective as it should be. Further it adds unnecessary costs such as wastage of money and time. Employees who use the 8.00 pm transport service are not satisfied with the service received, frequently complaining about the time they have to waste resulted by poor transportation planning. Yet there is no easy and straight forward solution, which can satisfy all the involving parties while giving the solution to vehicle routing problem. Thus, finding a solution for the issue becomes a motivation for this research which led to the learning about existing solutions, their usages and drawbacks.

1.4 Problem in Brief

Although there has been approaches made towards solving this issue there are minor setbacks which are still present. Finding the most cost effective solution for Vehicle Routing Problem with customer satisfaction is one of the major problems in the transportation and operation area. This issue will be analyzed and solved by the proposed solution. The chapters will provide the necessary content of the research together with the proposed solution.

1.5 Novel approach for Vehicle Routing Problem

Existing approaches for solving Vehicle Routing Problem are based on search algorithms and mathematical algorithms. These solutions theoretically give accurate results for specific scenarios of Vehicle Routing Problem within a set of predefined conditions. When it comes to real world problems, these have multiple involving parties with dynamically changing environments and a complex set of requirements from each individual who participate for Vehicle Routing Problem. This matter makes real world Vehicle Routing Problems more complex, making it harder to solve using traditional search and mathematical based solutions.

Multi Agent technology can be used to overcome this difficulty by enabling all stockholders as agents and giving negotiation power to influence the solution. The power of agent technology is based on the autonomy of agents, negotiations between them and finds solution as an emergent property. Application of such technology to the problem identified above, will give autonomy as well as best results. So the system is able to give efficient solutions consuming less time and cost. Since it cares about the all the aspects of the problem this will be a total solution for Vehicle Routing Problem.

1.6 Structure of the thesis

The rest of the thesis is structured in the following order. Chapter 2 critically reviews the domain of Vehicle Routing Problem by highlighting current solutions, practices, technologies, and limitations defining the research problem. Chapter 3 describes essentials of Multi Agent Technology showing its relevance to implement a solution for Vehicle Routing Problem. Chapter

4 presents the content of the Multi Agent Technology based approach to solve Vehicle Routing Problem. Chapter 5 discusses about the design of the proposed solution. Chapter 6 contains details of implementation of the system. Chapter 7 concludes the outcome of the research with the note on further work.

1.7 Summary

This chapter provides the complete picture of the whole project presenting the research problem, objectives, hypothesis and the novel solution. The following chapter will be based on the literature review of Vehicle Routing Problem in terms of practices, technologies and issues for defining the research problem.

Chapter 2
Current Issues and Practices in Vehicle Routing Problem

2.1 Introduction

In order to present a solution to the research topic, it is vital to identify the existing practices associated with Vehicle Routine Problem. Additionally, an insight about the analysis on the current issues that are present in Vehicle Routine Problem will create the foundation that is necessary to present the upcoming solution. The following chapter provides the content about the current issues and practices in Vehicle Routing Problem. Initially, it discusses about the history of Vehicle Routing Problem providing an insight of its inception and the attempts taken by the pioneers. Additionally, it will include the definition of the Vehicle Routing Problem and varieties of Vehicle Routing Problem. Later it will provide a discussion and critical reviews about the existing solutions and technologies for VRP. Finally it will describe current problems and limitations in explained techniques.

2.2 Vehicle Routing Problem – Current practice and issues

2.2.1 Vehicle Routing Problem

The Vehicle Routing Problem (VRP) has been extensively studied in optimization literature. VRP was introduced by George Dantzig and John Ramser in 1959 as a generalization of the well-known Travelling Salesman Problem (TSP) [1]. The specific paper was an attempt to solve the VRP using a linear programming formulation which was for obtaining a near optimal solution [1]. In 1964, it was continued by G. Clarke and J. R. Wright who tried to develop an iterative procedure which enables the rapid selection of an optimum or near-optimum route [2].

After late 60's many researchers became interested in the Vehicle Routing Problem which led them to come up with various solutions. These solutions can be divided into 4 main categories as Exact, Heuristics, Meta-heuristics and Hybrid methods [3]. Vehicle Routing Problem is divided in to several variations and specializations such as the Vehicle Routing Problem with Time Windows (VRPTW), Capacitated Vehicle Routing Problem (CVRP), Vehicle Routing Problem with Multiple Trips (VRPMT), Open Vehicle Routing Problem (OVRP) and Vehicle Routing Problem with Pickup and Delivery (VRPPD).

Currently VRP is widely studied because of its wide applicability and its importance in determining efficient strategies for reducing operational costs in distribution networks. For example, according to Geir Hasle, Knut-Andreas Lie and Ewald Quak [11] computer optimization programs can give savings of 5% to a company as transportation is usually a significant component of the cost of a product (10%). According to financial statistics the transportation sector makes up 10% of the GDP of the European Union. Consequently, any savings created by the VRP, even less than 5%, is a significant value. Giving a solution for the stated issue will result in saving a large amount of money which is vested without purpose.

2.2.2 Exact Algorithm Based Solutions

The branch-and-bound, the branch-and-cut and the branch-and-price are some exact algorithms which are proposed by researchers to solve VRP [3]. The Authors of [7] proposed an exact branch-and-price algorithm for solving the multiple vehicle routing problems with time windows. F.J.Bard, G. Kontoravdis and G. Yu have developed a branch-and-cut procedure for the VRPTW [6]. They addressed the problem of finding the minimum number of vehicles to visit a set of nodes subjected to time window and capacity constraints. The fleet is homogeneous and is located at a common depot. Each node requires the same type of service. An exact method is introduced based on branch-and-cut.

The objective of the VRPTW is to serve a number of customers within predefined time windows at a minimum cost (in terms of distance travelled), without violating the capacity and total trip time constraints for each vehicle. Combinatorial optimization problems of this kind are non-polynomial-hard (NP-hard) [4] and are hence best solved by using Heuristics. The most important Meta-heuristics used to solve the VRPTW are Tabu Search (TS), Genetic Algorithm (GA), Evolutionary Algorithms (EA) and Ant Colony Optimization Algorithm (ACO). Meta-heuristics controlling local search processes such as Tabu Search, simulated annealing genetic algorithms, evolution strategies, large neighborhood search, and guided local search [3]. Homberger and Gehring introduced a two-phase hybrid metaheuristic for the Vehicle Routing Problem with Time Windows (VRPTW) and a central depot [5]. In the stated solution they proposed to combine two methods which focus on minimization of the number of vehicles (primary criterion) and total travel distance (secondary criterion) [5]. To minimize the number of vehicles they used (μ,λ)-evolution strategy and to minimize the total travel distance Tabu Search algorithm was used. G. B. Alvarenga and team suggest a genetic and set partitioning two-phase approach for the VRPTW. The VRPTW is formulated as a set partitioning problem (SP). The GA is based on natural reproduction, selection and evolution of Darwin's theory. Ever since, GA has been popular on this problem domain because it can contribute to find better solutions for complex mathematical problems, such as the VRP and other NP-hard problems, in a reasonable amount of time.

P. Shaw from the University of Strathclyde proposed a local search method known as the Large Neighbourhood Search (LNS) for solving Vehicle Routing Problems [9]. The technique explores a large neighborhood of the current solution by selecting a number of customer visits to remove from the routing plan, and re-inserting these visits using a constraint-based tree search. M. Gandreau and team proposed a neighborhood search heuristics to optimize the planned routes of vehicles in a context where new requests, with a pick-up and a delivery location, occur in real-time [10]. This is a Dynamic Vehicle Routing Problem (DVRP). Within this framework, new solutions are explored through a neighborhood structure based on ejection chains.

All of the above existing researches suggest various methods to solve Vehicle Routing Problem. They use exact algorithms, heuristic methods, metahuristic methods and combination of these techniques (hybrid) as solution to Vehicle Routing Problem. These methodologies are based on search and mathematical algorithms and those algorithms are considering stakeholders of Vehicle Routing Problem as numerical or categorical data points with properties. When it comes to real world problems, those problems are having multiple involving parties with dynamically changing environments and complex set of requirements from each individual who participate for Vehicle Routing Problem. This makes real world Vehicle Routing Problems more complex and difficult to solve using traditional search and mathematical based solutions. Most of these solutions are focused merely to find a solution to the cost of the transportation and without concern for the uniqueness of the needs of other parties (client, vehicle owner, passenger …), who are part of the problem.

Multi Agent Technology can be used to overcome this difficulty by enabling all stakeholders as agents and giving negotiation power to influence the solution. The power of agent technology is based on autonomy of agents, negotiation between them and finds solution as an emergent property. Application of such technology to the problem identified above, will give autonomy as well as best results. So the system will give efficient solutions with less time and cost. As it cares about all the aspects of the problem this will be a total solution for Vehicle Routing Problem.

2.4 Summary

As the beginning chapter, this chapter explained about current practices in VRP simultaneously providing a critical review of these current practices by their highlighting issues. The details stated in the chapter provide the groundwork to understand the following chapters of the research. It will become the basis for the research and will be a reference to see how the Vehicle Routine Problem is solved with the Multi Agent System. In next chapter, it will discuss about Multi Agent Technology and how it is more suitable to solve VRP.

Chapter 3

Multi Agent Technology

3.1 Introduction

The previous chapter was based on how the Vehicle Routine Problem was solved through the current practices. The following chapter will be based on introducing the Multi Agent System, which can be used as a feasible method to solve the Vehicle Routine Problem. It will be further expanded with a discussion as to why the Multi Agent System is suitable to solve the VRP. For the purpose of comprehending the Multi Agent System, examples are discussed and applied in the manner that will show how it is applicable to solve the Vehicle Routine Problem. Additionally, a comparison is done to distinguish the differences between the traditional systems and the Multi Agent Systems. The key features of the Multi Agent system will provide the knowledge as to why it is more suitable to solve the VRP.

3.2 Multi Agent Technology

Multi Agent Technology is composed of multiple agents who possess the ability to pass messages to each other and achieve a certain goal. A single agent in a multi agent system is a small program that execute when required, do the assigned task and terminates [14]. While single agent does not make any sense, a collection of agents are capable of achieving many things. An Ant Colony is an example of a real world multi agent system, where each ant does a particular work to achieve a certain goal. While passing messages to other ants, they achieve this goal. In the software viewpoint, multi agent system is a collection of small programs that passes messages to each other to achieve a certain goal. For individual agent, that agent has a small knowledge, this little piece of knowledge is known as Ontology. The ontology may contain how the agent should act towards certain inputs from the environment. Unlike other Artificial Intelligence technologies, the software agent has an input from the environment, and the agent's reactions will change the environment. These agent systems have certain features such as autonomous, proactive and emergent associated with it.

Autonomous agent means that the agent does its work automatically without human intervention. The lifecycle of an agent (start, process and exit) is automatic. Proactive agent means that an agent does some sort of work related to achieving a desired goal when the agent is idle. Emergent feature (Collective Intelligence) is not associated with an individual agent; it's associated with the whole multi agent system. Emergent is the property where an idea or some suggestion suddenly pops up due to the communication among each other [14]. For example, a group of people discusses about buying a vehicle. At the beginning, they discuss about buying a car. Different people engage in a conversation about different brands, prices etc. however, at the end of the discussion they all decide to purchase motorcycles. In the above example, buying motorcycles is the emergent property.

3.3 Traditional Systems vs. Multi-Agent Systems

In particular, two large software systems, one traditional and the other agent-based are compared. The following table shows the differences between traditional systems and multi-agent systems.

Traditional systems	Multi-Agent systems
• Hierarchies of large programs • Sequential execution of operations • Instruction from top to bottom • Centralized decision • Data driven • Predictability • Stability • Striving to reduce the complexity • Total control	• Large networks of small agents • Parallel execution of operations • Negotiations • Distributed decisions • Knowledge driven • Self-organization • Evolution • Striving to thrive with the complexity • Support for growth

Table 3.1: Differences between Traditional systems and Multi-Agent systems

3.4 Key features

This following table shows key features and benefits of multi-agent systems together with usage examples for each feature.

No	Key Feature	Benefits	Usage examples
1.	Agent design supports full life cycle: perception, planning and execution.	Support of real time solutions and applications.	Dynamic dispatching, planning and optimization of mobile resources.
2.	Adaptive p2p networks of real-time schedulers demonstrating coevolution of self-organized systems.	Openness, high performance, scalability and reliability of enterprise-ready system.	Supply chain management. For example, factory and transport plans coordination.
3.	Dynamically formed bottom-up ontologies constantly renewed in the process of interaction with user.	Opportunity to fill the knowledge base on the fly without reprogramming.	In the process of interaction with a driver the system learns about events such as snowfall, road closure, etc.
4.	Emergence property. (Collective Intelligence)	Increase of business performance, efficiency, productivity and competitiveness.	A taxi driver using his mobile phone can pass signals about the groups of potential passengers.
5.	Sophisticated interaction with the system to find a solution to the problem.	Intellectualization of dialog with the user.	Allows the user to play with the solution and adjust it on the fly accordingly.

6.	Support of work in case of uncertainty or errors in initial data.	Lack of data or errors is not a restriction for the system to work.	Stability and reliability of results in case of incorrect data.
7.	Support of parallel processing.	Significant increase of performance.	Planning and scheduling of large amount of resources.
8.	Combination of work in real-time with classical batch algorithms of resource planning and scheduling.	Improvement of planning quality when orders and resource are known in advance.	Strategic long-term factory scheduling.

Table 3.2: Key features and benefits of Multi-Agent systems

3.5 Benefits of using MAT to solve Vehicle Routing Problem

As stated, current solutions for Vehicle Routing Problem are not satisfactory. To address this situation, Multi Agent Technology can be used. Due to the requirements and features that address each requirement Multi Agent Technology is the most suitable technology to solve the given problem.

- **Find optimal route**
  Main concern of Vehicle Routing Problem is to find the optimal route, which minimizes the traveling cost. There is a possibility to find the optimal route using features of multi agents such as Autonomy, Communication, Negotiation, Adaptation and Emergent property.

- **Considering needs of each individual**
  In multi-agent systems each agent can have their own Beliefs, Desires, and Intentions (BDI). Therefore each individual can be represented by an agent and they will negotiate with other agents to satisfy their needs, while contributing to create the solution.

- **Finding and sharing required knowledge**
  Each agent has their own ontology and they can access the shared ontology as well. Since agents are autonomous they can be programmed to find the knowledge they need. Through the communication they can share knowledge and they can create a new knowledge which the individual members do not have (emergent property). This will help to improve the quality of the solution.

3.6 Summary

In this chapter, the discussion was based on Multi Agent Technology. The discussion contained features of the software agents. Furthermore, it portrayed how this technology is capable to solve Vehicle Routing Problem more efficiently as opposed to the traditional systems that have been used in practice in the current VRP context. The benefits of the MAT were listed to provide an insight to the system. In next chapter, the discussion will be based on the approach to solve Vehicle Routing Problem.

Chapter 4

Solve Vehicle Routing Problem using Multi Agent Technology

4.1 Introduction

The previous two chapters defined the research problem: the inefficiency and incompetency in current approaches to solve Vehicle Routing Problem. Furthermore, the discussion was extended as to why the Multi Agent Technology is the potential technology to develop novel method to solve VRP. With a clear understanding of the VRP and the MAT, the next approach would be how the two can be connected to bring out a solution. This chapter is based on the approach which is suggested by the proposed system to solve the VRP. Based on the hypothesis, the system input, system output and the process are discussed with the non-functional requirements of the system. Furthermore, the features of the system are explained. At the end of the chapter and evaluation is stated through comparison and justifying the thesis.

4.2 Hypothesis

Vehicle routing problem be can solved by the Multi Agent systems.

4.3 Input

Acquiring required information from environment is one of the main features in Multi Agent Technology. For agent creation process, system should know passenger details and vehicle details. In negotiation process passenger agents and vehicle agents should have ontology with required information. Following are the inputs, which collect through user inputs and integrated systems.

- Destination Points Details
- Vehicle Information

- Constraint Values
- Route Information

Destination points will contains details regarding passenger drop off points and passenger details. Properties of destination point include passenger name, latitude of the drop off point and longitude of the drop off point. The system will collect the vehicle name (unique ID) and number of seats of the vehicle to create vehicle agents. The number of vehicles is a constraint variable for this system. Agents of the system can interact with Google API to get route information that is required. As an example, if passenger agents need to know the estimated travel time from departure location to the drop off location on 8.00 pm traffic conditions, it can directly call Google API and acquire the required information. Input sources can be divided into two main categories. One is User Inputs (Destination Points, Vehicle Information and Constraint Values) and other one is Web Services (Route Information using Google API). Using both these input types, the system will generate a solution for Vehicle Routing Problem.

This system runs based on two modes,

- Administrator Mode
- Normal Mode

For Administrator Mode, user input to the system should be added to the system by the user of the system. In the Normal Mode, the system will take all the information required through integrated systems, where this user has already added their requests.

4.4 Output

Output of this system will be routes which design to minimize the cost and maximize passenger satisfaction. Output format and representation is playing a major role in any solution. It affects how user derives information and how user interpreted the solution. For this system output is representing in two formats.

- Geographical map
- Table of representing how passengers assign to vehicles

Because of this solution is providing a route plan it is good to use geographical map as one of outputs, which help user to understand overall plan at a glans. Furthermore it is using a table to represent how vehicles allocated for passengers. This will help to share information among stakeholders with clear meaning.

4.5 Process

By using passenger details, the system will generate a passenger agent for each destination point while setting a passenger name as passenger identification name. When passenger agent comes to life, it will connect with Google API and retrieve minimum travel duration estimation from departure location to drop off location with 8.00 pm traffic condition. Based on minimum travel duration estimation it will generate maximum travel time threshold value for itself, which is 20% addition to minimum travel duration.

i.e. If passenger no: 1's minimum travel duration estimation = 100 min
Then his generate maximum travel time threshold value = 100 + (100* 20/100)
= 120 min

Using a percentage of minimum travel time estimation as the buffer for maximum threshold value helps to generalize user specific threshold values based on distance between departure locations to passenger drop off location.

Vehicle agents will be created based on vehicle name and number of seats that are included. After passenger agents are created each passenger agent broadcasts a message with its location details (longitude and latitude) to all other passenger agents to identify nearby agents. When a passenger agent receives a nearby request message from another passenger agent it will connect to Google API and based on their location coordinates it will get travel time duration estimation between them. If that duration is less than 30% (this percentage can be configured) of its minimum travel time, receiver agent adds requested passenger agent as its nearby agent. Passenger agents identify nearby agents and create groups using this process. After the creation of groups for each group, group agent comes to life and it starts to negotiate with vehicle agents on behalf of its members.

When a vehicle agent receives a request from a passenger agent or group agent, first it checks whether it has the required number of seats or not. If the required number of seats is not available, it sends a message to request agent informing that the capacity is not enough. If the required number of seats is available, it checks whether adding this agent or group will affect badly for existing passengers. If it does not affect accepting passengers, the vehicle agent sends the minimum travel time it can allocate for the requested passenger or group. Then the passenger or group agent collects all the responses and selects the proposal which has minimum travel time and checks it against his maximum travel time threshold. If it is satisfied by selected proposal, passenger agent sends acknowledgment to the vehicle agent. Using this process passenger agents, group agents and vehicle agents negotiate among each other and creates a solution as an emergent property.

This process can be summarized as following. Passengers and vehicles assigned to agents and passenger agents create group agents by grouping with nearby agents. Through the negotiation process vehicles choose their passengers and routes (passenger agents will try to get their maximum comfort and vehicles will try to reach minimum distance). A solution is generated as an emergent property of that negotiation and that solution satisfies requirements of all involving parties while being cost effective.

4.6 Features

Being Vehicle Routing Problem a NP-hard problem makes it is difficult to use exact algorithms when it has large number of problem instances. Because of Multi Agent Technology features this system will facilitates following features which are most suitable for solving Vehicle Routing Problem.

- Complexity of the problem will be handled by MAS.
  Multi Agent Systems are not ruled top-down by centralized system control. Instead, complexity is distributed across simplified agents, which are capable to interact with each other and provide emergent solutions.

- Adaptation to dynamic situations
  Due to the de-centralized and emergent solution behavior, this system will adapt to the situation and find the best solution based on it.

- Autonomy of the system
  This system will find solutions without intervention of the user.

4.7 Users

People who need to design routes for transportations will be the users of this system.

4.8 Evaluation

Comparison of the system provided solutions against manually generated (existing) solutions for a selected data set. If solutions generated by MAS are cost effective against manually created solutions, the hypothesis is acceptable.

4.9 Summary

This chapter was based on the approach of the project. It included a discussion about the hypothesis of the research together with the input, process, the output process, users and the evaluation. The features that the system facilitates were also added in order to provide a detailed analysis on the approach. It further justifies the reason as to why the system was specifically chosen as a solution. The next chapter will describe about the design of the system.

Chapter 5

Design of Multi Agent System to Solve VRP

5.1 Introduction

It is vital to have a clear design that would enable to use the Multi Agent System efficiently to solve the Vehicle Routine Problem. The following chapter will showcase the design that has been implemented for the mentioned purpose. The system will focus on a specific problem; such it will need to find the most cost effective routes for dynamically allocated passengers at a given period of time. Vehicles' details and passengers' details are provided for the benefit of having a clear approach. Furthermore, there is a constraint on the maximum ride time of each passenger, which should not be exceeded. The medium of communication used by the agents is defined. Additionally, each role of each agents of the system is separately defined for the purpose of understanding their importance in the system.

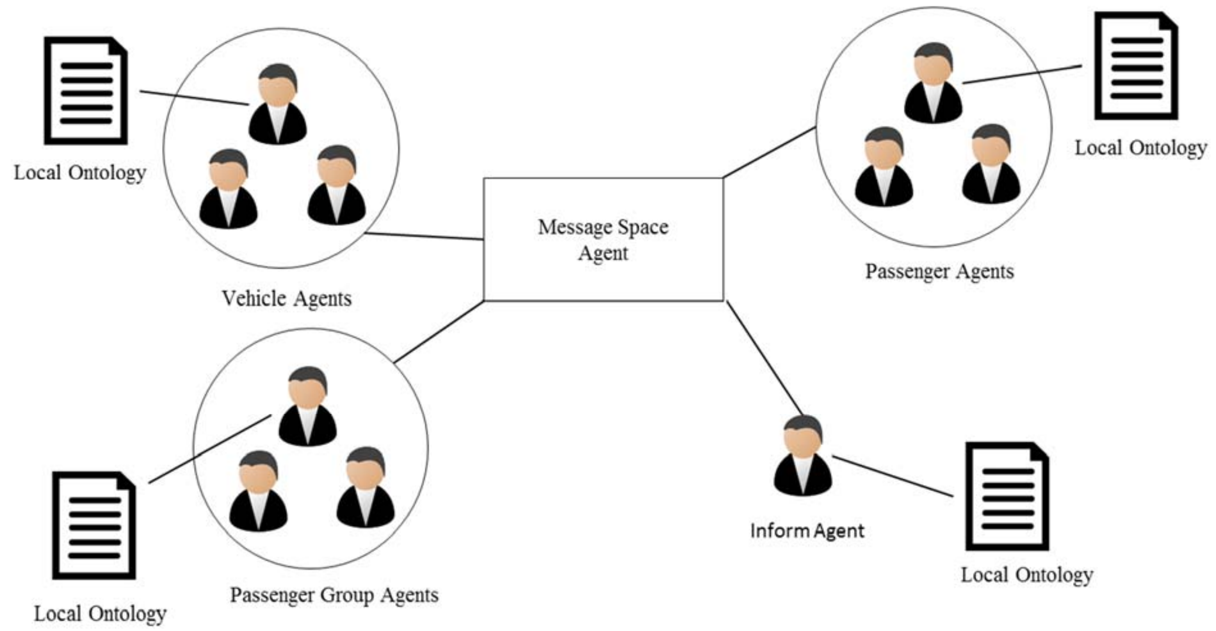


Figure 5.1: Design of the system

The system is designed as such that it will automatically generate an agent for each individual stakeholder (passenger, vehicle) to represent their requirements. Passenger agents are designed to request vehicles by giving their information to other agents and to negotiate for a minimum ride time for them. Vehicle agents are designed to negotiate for most cost effective routes without exceeding their capacity. Likewise passenger agents will negotiate for quality of the service while vehicle agents will negotiate for cost effectiveness of the solution. Therefore, the system will be able to find a cost effective solution while providing required level of service quality to the passenger.

5.2 Message Space Agent

The field of Multi-Agent Systems (MAS) is driven by the key metaphor of interaction between multiple autonomous agents whose micro-behavior produces the perceived over-all system behavior. Therefore Communicating and sharing intelligence among agents is an important facet. To fulfill this requirement message space plays a major role in Multi Agent Technology.

Message space acts as a middle man for all the communication between agents. Request agents and the resource agents are connected to the message space and can be identified using directory service. If one agent needs to communicate with another agent, then that agent connects to message space and sends the message to the required agent. Afterwards, message space searches that agent using directory service and route that message to the relevant agent. Additionally, by using message space an agent can broadcast a message to multiple agents. It is also responsible for coordination of the user channel request.

5.3 Passenger Agents

These agents will be created automatically by using passenger specific information. Each passenger agent will have details regarding their destination (longitude, latitude) and a unique name to identify. At creation time a passenger agent uses his or her destination details and call Google Map API web services to capture the minimum travel time needed to reach home based on 8.00 pm traffic. By adding 20% of expected minimum travel time to itself, it generates a maximum expected travel time threshold value.

At the start passenger agents will talk with other passenger agents to find nearby passengers. If one passenger finds a nearby agent they will create a group. If an agent finds another group nearby, the agent will join with that group. Likewise passenger agents will form into groups and negotiate with vehicles as a group, where they can reach their destinations with lowest travel time. If a group or single agent finds a vehicle which satisfies their requirements, they will choose the specific vehicle.

5.4 Vehicle Agents

The system creates vehicle agents based on vehicle information (vehicle name and number of seats of each vehicle) which provide as input to the system. When a vehicle agent receives a message from a passenger agent or group agent, it checks whether the required capacity is available or not. If the capacity is not available, vehicle agent rejects the request and informs passenger agent that the requested capacity is not available. If the required capacity is available and vehicle has some passengers already assigned, it connects with Google API to get estimated travel time for requested passenger while considering passengers that are already assigned and it checks whether adding this passenger will affect negatively for already assigned passenger requirements. If adding requested passenger is not affect negatively for assigned passengers, it sends an estimated travel time as a proposal for passenger. Else it rejects the request and informs the passenger agent.

When passenger agent receives all the proposed estimated travel time, it selects the proposal which contains the minimum estimated travel time, accepts that proposal and informs the vehicle agent to assign him as a passenger of that vehicle. It rejects all other vehicle agents' proposals and informs them. By using this process vehicle agents are negotiating with passenger agents to design more cost effective routes while satisfying the passenger agents. At the same time they are negotiating among themselves to separate and allocate more cost effective passengers for their own routes.

5.5 Inform Agent

This agent will load at the startup and its main responsibility is to inform passenger agents to perform groups based on their relative positions after make sure all the passenger agents are live. After passenger agents are created, it broadcast message to passenger agents for communicate with one another and create groups. After giving this message, it will sleep till passenger agents create their groups and passenger group agents come to live. When that process completed, it broadcast messages for passenger agents and vehicle agents to start the negotiation process for generating solution.

5.6 Group Agent

Every time a passenger agent connects with a nearby agent it will create a new group agent and add both passenger agents into the created group or it will add passenger agents into existing groups. After group agent comes to life, member passenger agents will assign group agent to negotiate on behalf of them and stop direct negotiations with vehicle agents. Group agents can join with other group agents or passenger agents and create new groups. Any group agent has two or many passenger agents and those passenger agents are sorted based on the distance from departure location.

The passenger who has minimum expected travel duration and the passenger who has maximum expected travel duration among other agents are explicitly captured in group agent ontology. When new an agent joins the group these two records are updated based on joining passenger agent. When group agent evaluates a proposal of vehicle agent, first it checks whether this proposal satisfies the nearest member agent's (who has minimum expected travel time) requirement. If it satisfies then it checks the proposal against the requirement of the agent who has maximum expected travel time. If it satisfies the request group the agent will accept that proposal and ask the vehicle agent to assign them to the vehicle.

Group agent’s process and lifetime can be summarized as follows. Group agents’ created by two or more nearby passenger agents and it can be expanded by adding new nearby agents. Group agents will contain all member agents’ details (destination points) and their requirements (required travel times). When vehicle agent gives a proposal, this agent will evaluate it against member requirements. If the proposal satisfies member requirements, group agents will decide to use that vehicle and assign member agents to proposed vehicle.

5.8 Summary

The high level design of the system was explained in the chapter. The roles of the various agents in the system and their responsibilities were analyzed in detail so that in the following chapter it will be easier to locate these agents with the process. Next chapter is about the implementation of the Multi Agent Technology based solution, the technologies that are used and how each process is implemented.

Chapter 6

Implementation of Multi Agent System to Solve VRP

6.1 Introduction

The implementation of the Multi Agent System will be discussed in the following chapter. Under implementation, it describes how this system is developed. Furthermore it gives a detailed insight about the technologies, methods, algorithms that are been used. The System Implementation Design is included together with the process. The details of the process are given also through images which makes it easier to understand. All the steps of the process are been explained in an order to understand how the system works.

6.2 Technologies used

Selecting correct technologies for a system is a deciding factor for its success. Selecting an unsuitable technologies or incompatible set of technologies together to implement solution will make implementation process hard and even sometimes researcher has to pay the price by giving up the research. Therefore technologies for this system choose carefully with clear understanding about inputs, process and outputs to be generated. Jade framework is used as the core framework for this system and to support it Java is selected as underlying technology to run Jade framework on it. Likewise all the following technologies were used for implementation of the system.

- Java JDK 1.8.20
- JADE
- JavaFX
- Google Maps API
- HTML and Javascript
- Maven

6.2.1 Java JDK 1.8.20

JADE is one of the popular agent development frameworks and it uses java technology as its underlying technology. Since JADE is used as the agent framework of the present context; this system uses Java as its base technology. The Java Development Kit (JDK) is a software development environment used for developing Java applications. It includes the Java Runtime Environment (JRE), an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development. For this system JAVA is used as the base technology and JDK 1.8.20 as the Development Kit.

6.2.2 JADE

JADE (Java Agent Development Framework) is a software Framework fully implemented in the Java language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that support the debugging and deployment phases.

6.2.3 JavaFX

JavaFX is a software platform for creating and delivering desktop applications, as well as rich internet applications (RIAs) that can run across a wide variety of devices. It is intended to replace Swing as the standard GUI library for Java SE. Since it has rich features than other JAVA GUI frameworks, this system's Graphical User Interfaces (GUI) are developed using JavaFX.

6.2.4 Google Maps API

Google launched the Google Maps API in June 2005 as a free service to allow developers to integrate Google Maps into their websites. By using the Google Maps API, it is possible to embed Google Maps site into an external website, on to which site specific data can be overlaid.

This API communication is based on JavaScript and JSON and it is a service for retrieving static map images, and web services for performing geocoding and generating driving directions. In this application Google Maps API is used to display passenger destinations, display passenger groups, display routes, retrieve travel time for passengers and retrieve routes according to traffic.

6.2.5 HTML and Javascript

This application uses HTML to display geo map, passenger details, groups and routes. HTML helps to run Javascripts on it and connect with Google maps to show update geo map insight. To mark passenger destinations, groups and routes in HTML, application uses Javascript. Other than that Javascript is used as the communication technology to communicate with Google Maps API to acquire required knowledge for agents.

6.3 System Implementation Design

The system implementation design is presented as follows.

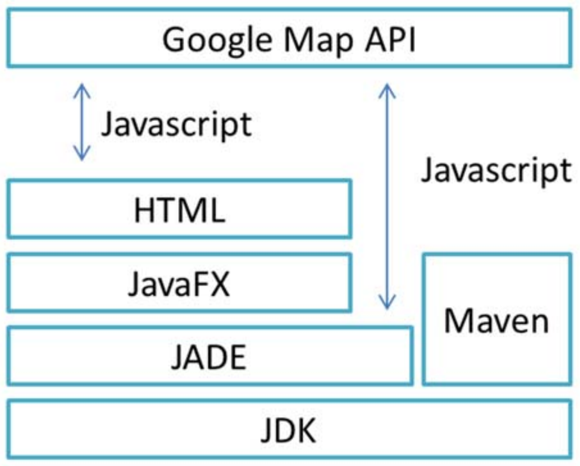


Figure 6.1: System Implementation Design

6.4 Process

When considering the whole process of the system, it can be divided it into three main phases. These phases are data acquisition and create agent phase, passenger agent grouping phase, passenger, passenger group and vehicle negotiation phase.

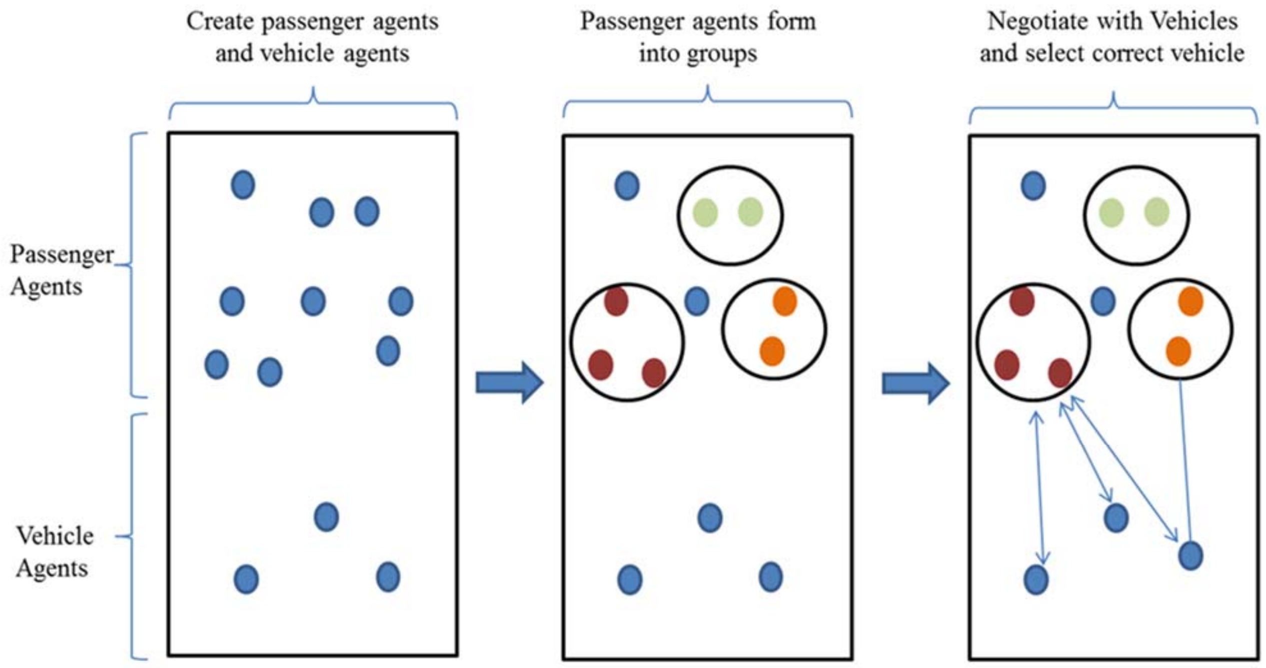

Create passenger agents and vehicle agents
Passenger agents form into groups
Negotiate with Vehicles and select correct vehicle
Passenger Agents
Vehicle Agents

Figure 6.2: Steps of the system process

6.5 Create Agents

When the system starts it will take passenger details of those who requested transport and vehicle details which are available at the given period of time. These details can be user input or an input coming from integrated system based on the mode the system is running on. To handle the life cycle of the agents, system uses Jade framework, which the most popular JAVA language based Multi Agent Technology framework with FIPA specifications compliant.

When the system receives the required information, it creates passenger agents, vehicle agents and informs agents according to it. After passenger agents start, they will call Google Maps API to find the minimum travel time to their destination based on traffic at travelling time and set it as their minimum expected travel time. By adding 20% of minimum expected travel time to itself they will find maximum expected travel time and they will negotiate with vehicle agents to have a travel time below that threshold value. Vehicle agents are created based on number of vehicles, vehicle names and capacity of vehicles.

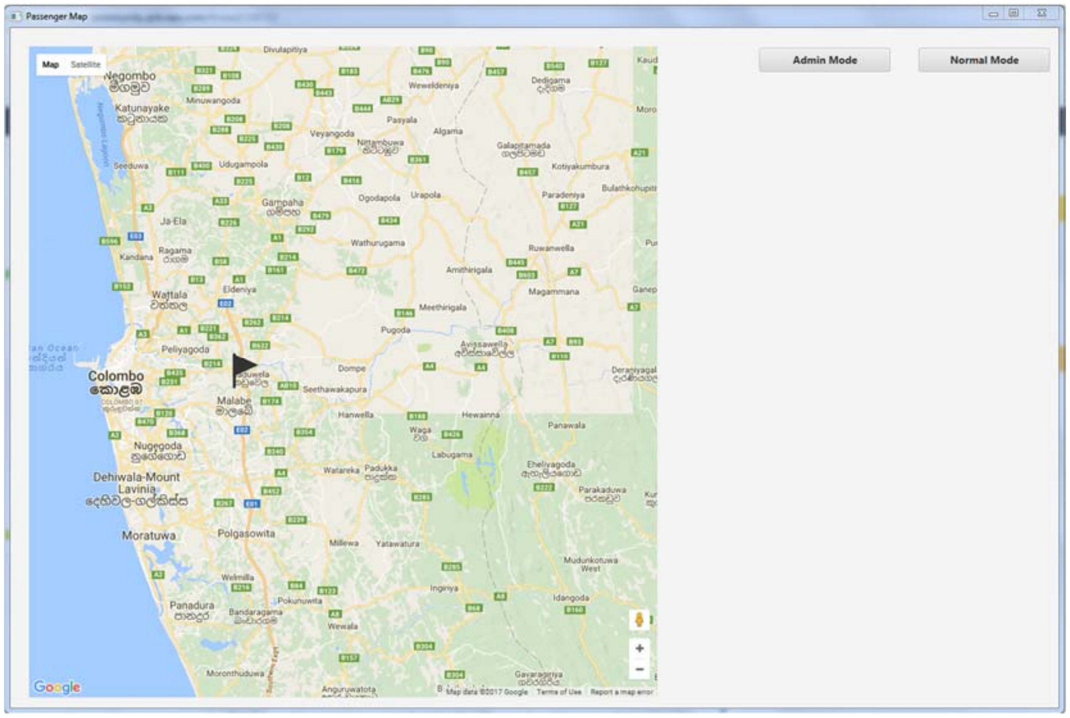


Figure 6.3: Initial User Interface– Normal Mode

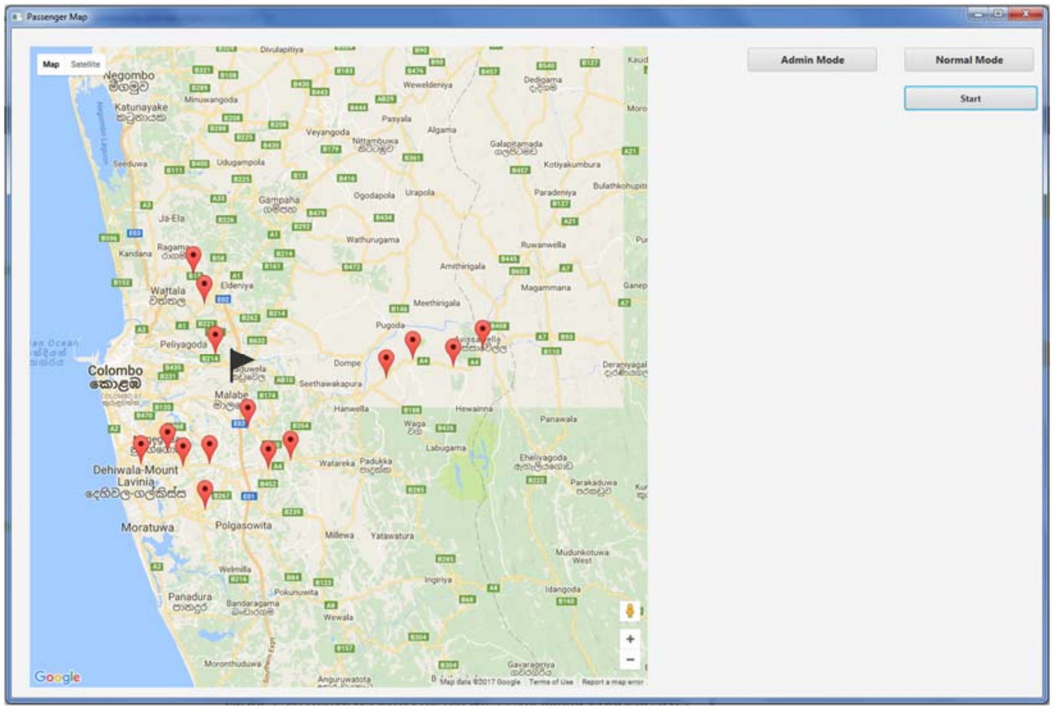
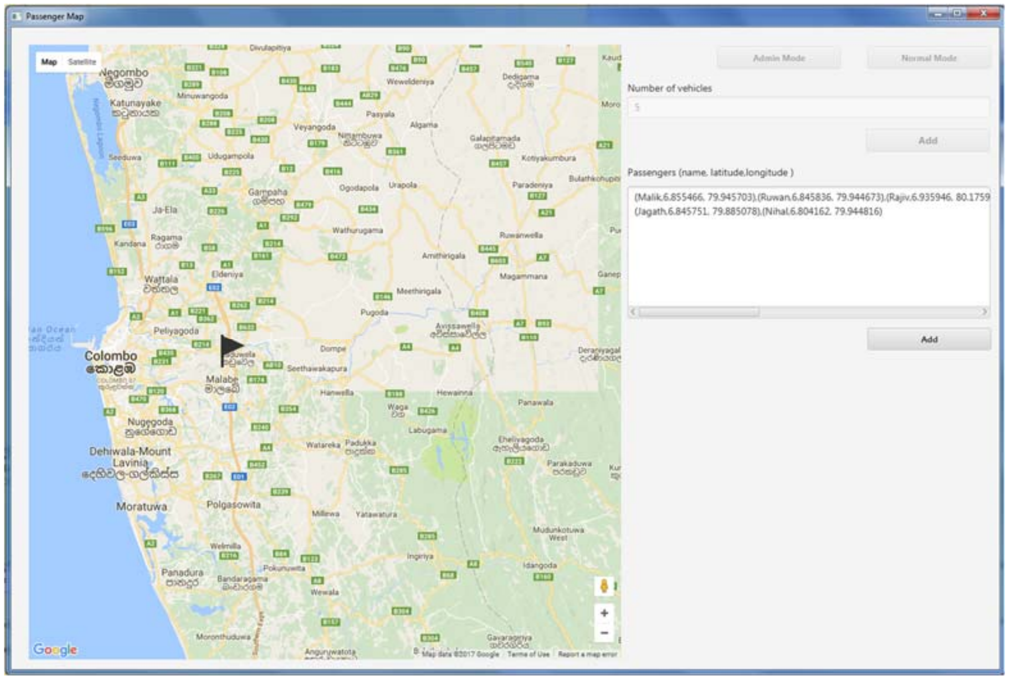


Figure 6.4: After Agent Creation – Normal Mode



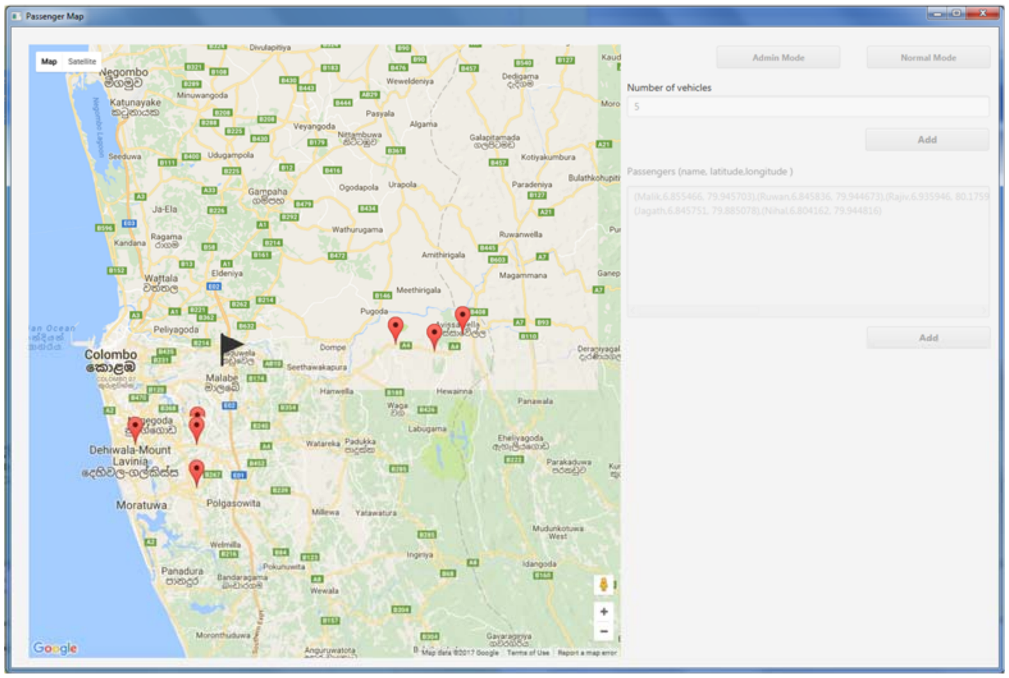
Number of vehicles

5

Passengers (name, latitude,longitude)

```
(Malik,6.855466, 79.945703),(Ruwan,6.845836, 79.944673),(Rajiv,6.935946, 80.1759
(Jagath,6.845751, 79.885078),(Nihal,6.804162, 79.944816)
```

Figure 6.5: Add Passenger Details – Administrator Mode



Number of vehicles

5

Passengers (name, latitude,longitude)

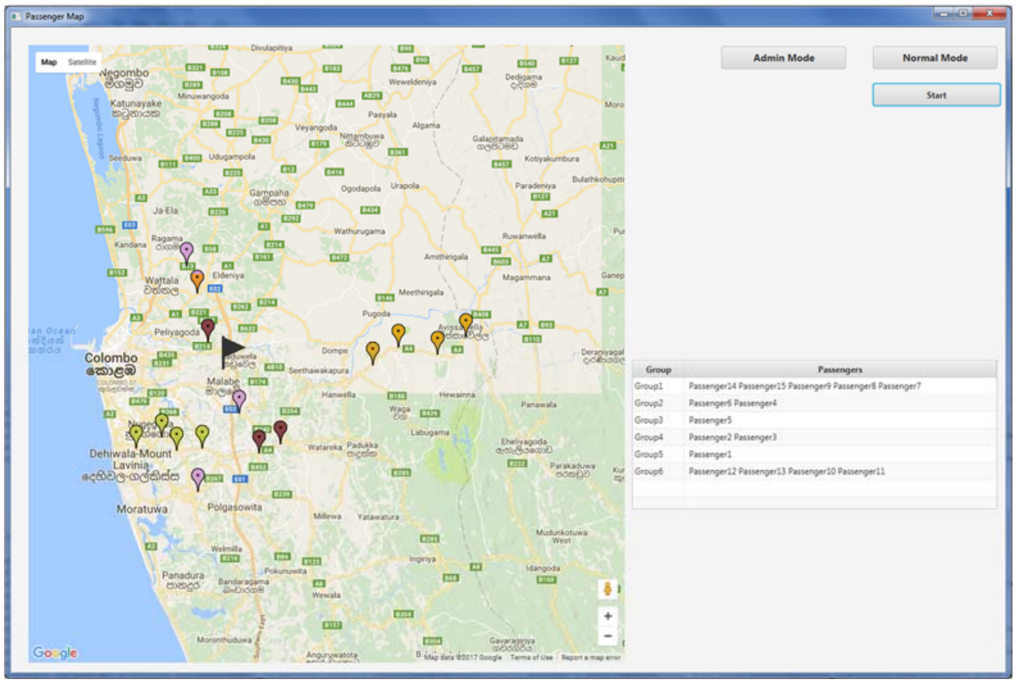
```
(Malik,6.855466, 79.945703),(Ruwan,6.845836, 79.944673),(Rajiv,6.935946, 80.1759
(Jagath,6.845751, 79.885078),(Nihal,6.804162, 79.944816)
```

Figure 6.6: After Agent Creation – Administrator Mode

6.6 Passenger agents form into groups

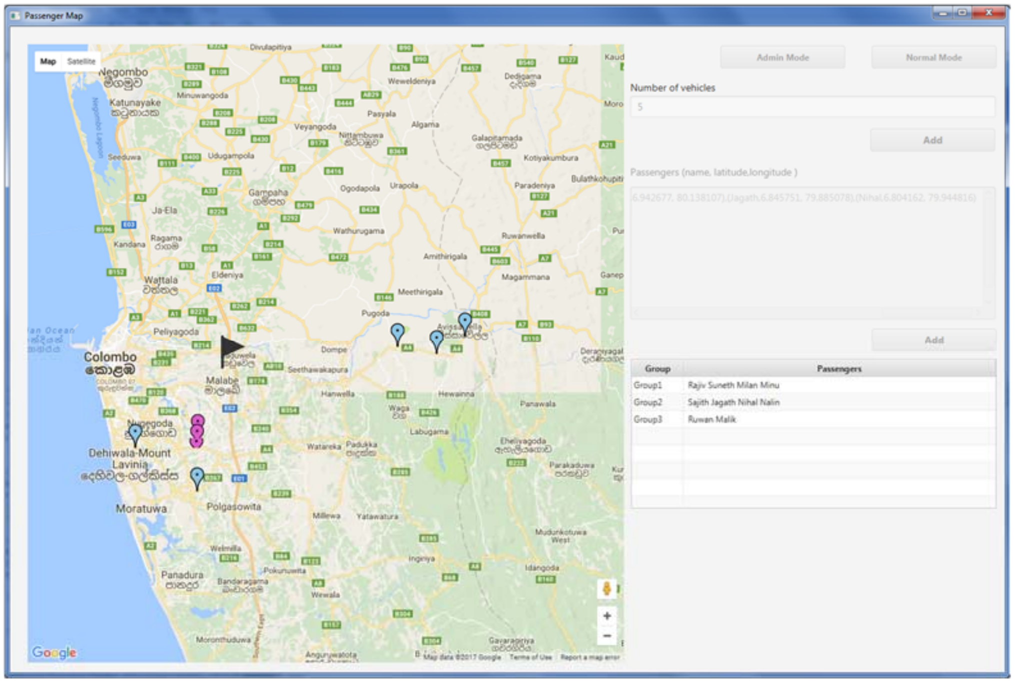
After passenger agents set their threshold values, inform agent will inform them to create groups by connecting with nearby passengers. Afterwards all the passengers broadcast a request message with their longitude and latitude details to identify nearby passenger agents. Later the agent who received the message gets the distance to request agent based on received details. If requested agent is within 30% of minimum expected travel time radius, it sends an acknowledgment message to the requested agent.

By using this process passenger agents are able to talk with each other and create groups with passengers who are within 30% travel time from their minimum expected travel time. After all the passengers are grouped into nearby groups, the system shows nearby groups details on a map by coloring them with the same colors. When passenger agents create a new group or join to a group they will be assigned to a group agent and the specific agent will take responsibility of finding a suitable vehicle for assigned passengers.



Group	Passengers
Group1	Passenger14 Passenger15 Passenger9 Passenger8 Passenger7
Group2	Passenger6 Passenger4
Group3	Passenger5
Group4	Passenger2 Passenger3
Group5	Passenger1
Group6	Passenger12 Passenger13 Passenger10 Passenger11

Figure 6.7: After passenger agents form into groups– Normal Mode



Number of vehicles

5

Passengers (name, latitude,longitude)

6.942677, 80.138107),(Jagath,6.845751, 79.885078),(Nihal,6.804162, 79.944816)

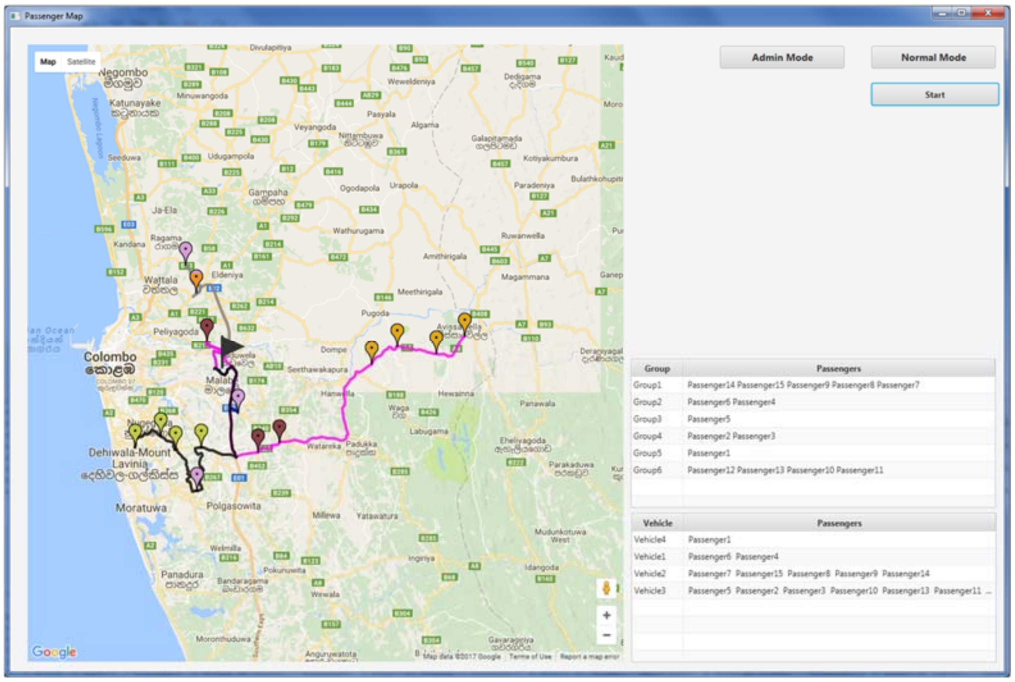
Group	Passengers
Group1	Rajiv Suneth Milan Minu
Group2	Sajith Jagath Nihal Nalin
Group3	Ruwan Malik

Figure 6.8: After passenger agents form into groups – Administrator Mode

6.7 Negotiation process

After the passenger agent is grouped into nearby groups, negotiation starts between passenger, passenger group and vehicle agents.

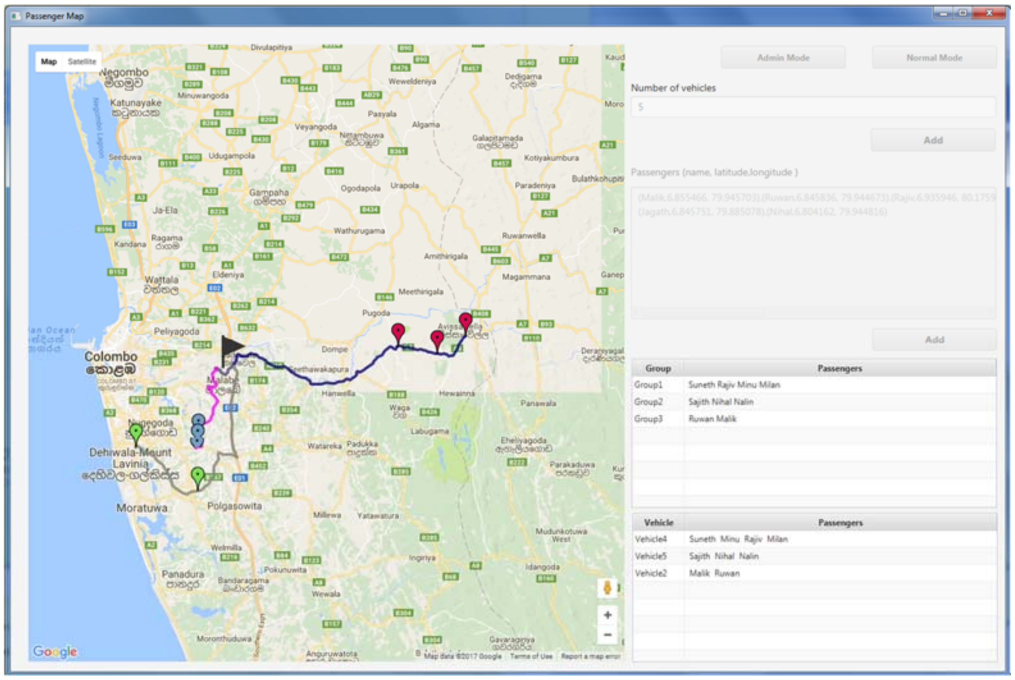
Passenger agents will try to negotiate for the lowest travel time they can have, while vehicle agents negotiate for the minimum route cost. At the end of this process both the parties will come up with a solution, which have minimum route cost while satisfying the passenger requirements.



Group	Passengers
Group1	Passenger14 Passenger15 Passenger9 Passenger8 Passenger7
Group2	Passenger6 Passenger4
Group3	Passenger5
Group4	Passenger2 Passenger3
Group5	Passenger1
Group6	Passenger12 Passenger13 Passenger10 Passenger11

Vehicle	Passengers
Vehicle4	Passenger1
Vehicle1	Passenger6 Passenger4
Vehicle2	Passenger7 Passenger15 Passenger8 Passenger9 Passenger14
Vehicle3	Passenger5 Passenger2 Passenger3 Passenger10 Passenger13 Passenger11 ...

Figure 6.9: After negotiation process– Normal Mode



Number of vehicles

5

Passengers (name, latitude,longitude)

(Malik,6.855466, 79.945703),(Ruwan,6.845836, 79.944673),(Rajiv,6.935946, 80.1759
(Jagath,6.845751, 79.885078),(Nihal,6.804162, 79.944816)

Group	Passengers
Group1	Suneth Rajiv Minu Milan
Group2	Sajith Nihal Nalin
Group3	Ruwan Malik

Vehicle	Passengers
Vehicle4	Suneth Minu Rajiv Milan
Vehicle5	Sajith Nihal Nalin
Vehicle2	Malik Ruwan

Figure 6.10: After negotiation process– Administrator Mode

6.8 Summary

This chapter discussed about the technologies used and the implementation of the system. Later it discussed about how passenger agents acquire required knowledge, create groups and how they negotiate with vehicle agents. Under that topic it described how passenger and vehicle agents take decisions for an uninterrupted flow of the service. Finally the discussion is extended to show how this process is presented to user of the system. The next chapter will explain an overall evaluation process of the research.

Chapter 7

Evaluation

7.1 Introduction

An experiment is designed to evaluate the Hypothesis in the given chapter. It will showcase efficiency and will reveal comparison of data that were being used. The experiment is centered on a particular data. Furthermore, it is justified as to how the data was selected to provide an unbiased experiment. The discussion is extended to see whether the objectives mentioned in earlier chapters are met and to what extent.

7.2 Experimental Design

This experiment was designed to evaluate whether shuttle plans generated by the system is efficient than manually generated shuttle plans or not. As the first step of the evaluation process sample dataset were selected from shuttle requests, which were made by the employees of MillenniumIT for 8.00pm shuttle. To select a dataset without any biasness, shuttle request data for 10 days was selected randomly.

Later for the evaluation process manually generated shuttle plans for those shuttle requests were collected and the data was summarized. Summarization was done based on the following criteria,

- Number of passengers
- Total route distance (for all requests)

Date	Number of passengers	Total route distance(Manually)
14/09/2016	43	503
16/09/2016	34	482
2009/2016	51	529
22/09/2016	12	358
25/09/2016	27	392
29/09/2016	32	453

30/09/2016	18	316
04/10/2016	45	539
06/10/2016	36	453
12/10/2016	17	337

Table 7.1: Sample data summary

After collecting and summarizing the dataset, shuttle plans were generated through the proposed system for the same requests. When shuttle plans were generated, the total route cost of each plan were calculated and evaluated against the manually generate shuttle plans. The side by side comparison of the two methods was as follows.

Date	Total route distance(Manually)	Total route distance (MAS)
14/09/2016	503	492
16/09/2016	482	476
20/09/2016	529	545
22/09/2016	358	332
25/09/2016	392	408
29/09/2016	453	412
30/09/2016	316	289
04/10/2016	539	513
06/10/2016	453	412
12/10/2016	337	325

Table 7.2: Route distance comparison

The percentage of the system generated shuttle plans was better than manually generated plans which were 80%. Only two sample datasets have total route distance lower than the solution generated by Multi Agent Technology. When compare route costs most of the system generated routes were efficient more than 5% with regard to the existing system.

7.3 Evaluate Results

Analyzing and evaluate results of system generated solution against the manually generated solutions were given concrete evidence to suggest that the system generated solutions were more cost effective than the manually generated solutions. As per the following diagrams, solutions were evaluated and come up with more insights about the solutions given by two parties.

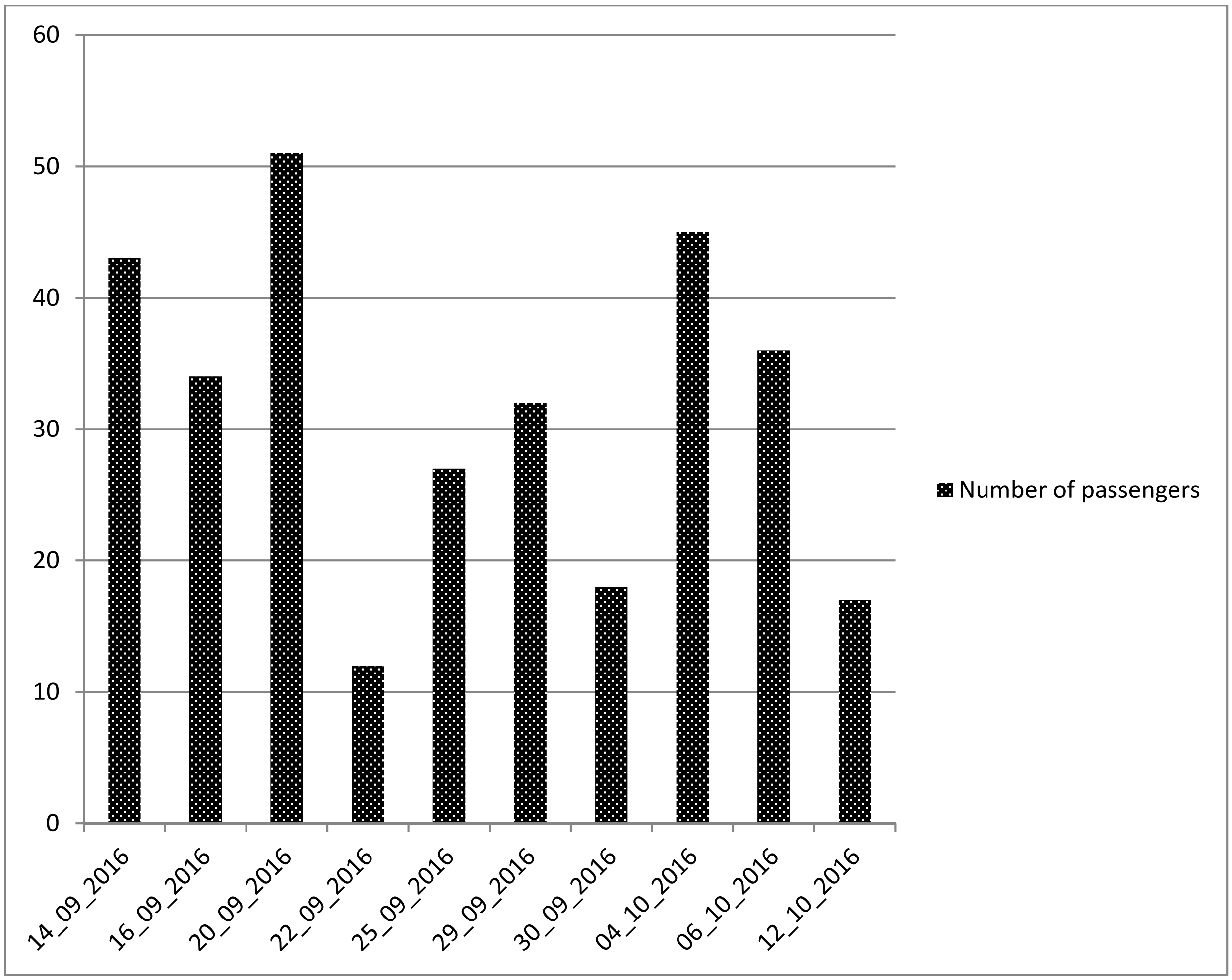


Figure 7.1: Number of passenger based on the date

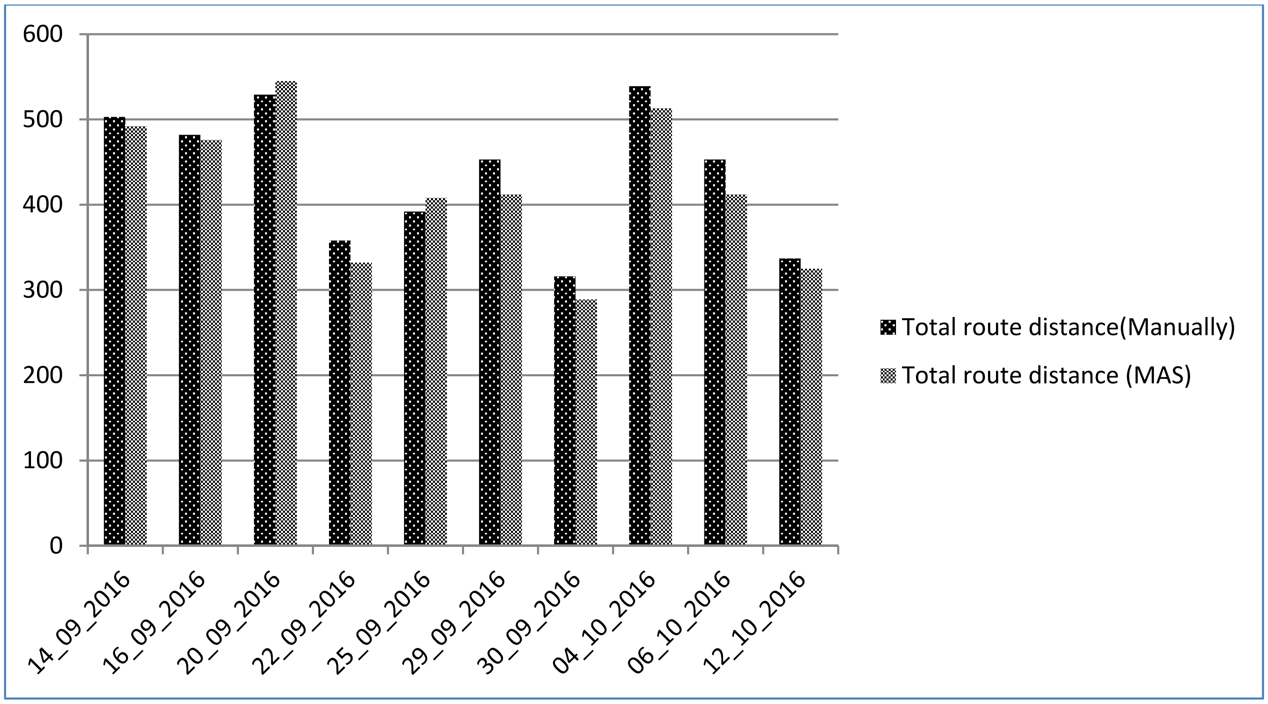


Figure 7.2: Comparison of two solutions

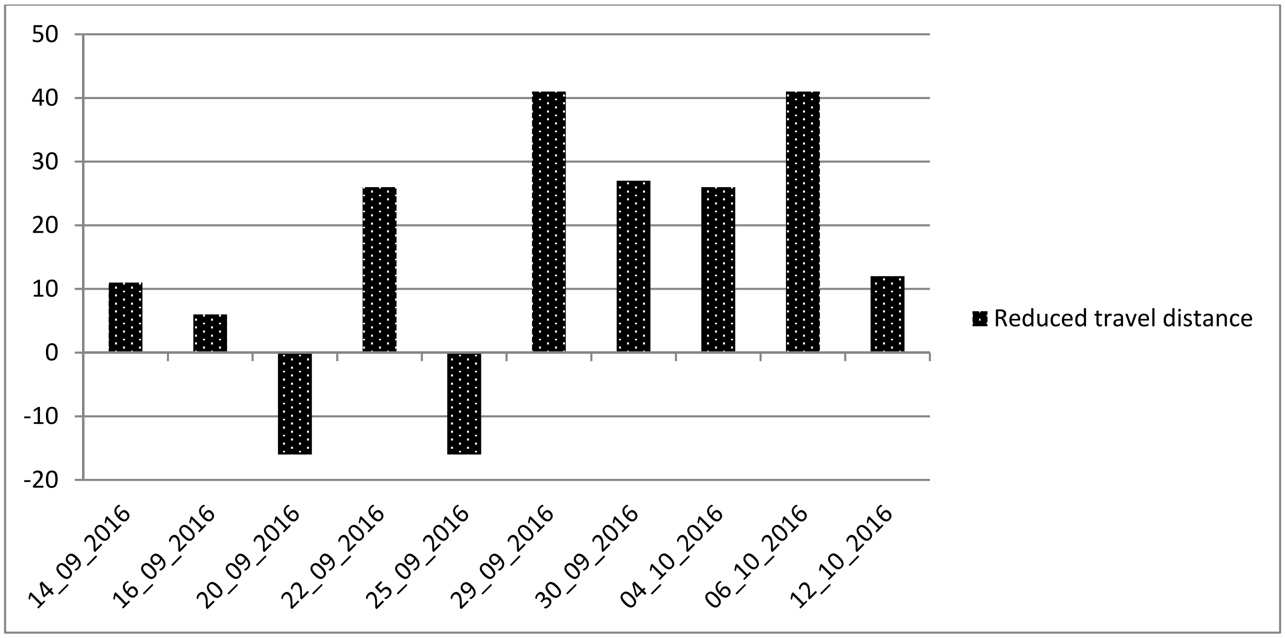


Figure 7.3: Total kms saving by system generated solution

As per the Figure 7.1 and Figure 7.2 it shows clear relationship between number of passengers and the total distances of generated solutions. It indicates solutions are creating based on the passengers requirements. All above insights suggest that system generated solutions are cost effective than the manually generated solutions for 80% of the times and it helps to reduce the transport cost by considerable amount.

7.3 Summary

In the end of the chapter, it brings out a detailed evaluation of the Hypothesis using a proper experiment mechanism. It has been designed and conducted in such away to measure the success of the application. All the attempts made to conduct the experiment are presented step by step in a detailed manner. The efficiency of the system is finally stated as opposed to the manual system. By presenting the results, the hypothesis is justified.

Chapter 8
Conclusion

8.1 Introduction

Through the previous chapters detailed discussions were done on the main problem, different kinds of technology that have been used, a hypothesis, design of the solution, implementation of the solution and its evaluation process. As the final chapter, this chapter explains the conclusion of the research. Additionally, it will explain the further work done in the proposed system. It describes the challenges that were found during the process and suggests what areas should be tented to, to run the system without any errors. However, the current improvements that are identified in the system are brought to a conclusion.

8.2 Conclusion

The aim of this project was to develop a Multi Agent System for generate travel plan, which is cost effective than the manually generated travel plan. Furthermore it should able to come up with solutions which satisfy all the involving parties. This is a challenging target for traditional technologies and mathematical based technologies, Because of the complexity it adds up to the solution with increase of the number of travel requests. Multi Agent Technology overcome this challenge by converting traditional data points to agents, who can have their own goals, negotiate for themselves and contribute for best possible overall solution.

Because of being a Multi Agent System this system can operate using minimal user inputs and intervention. It uses external systems like Google API to quire required knowledge and through the interaction and negotiation it builds the solution as an emergent property. Based on these features it has higher level of adaptation to different problems and it increases the level of satisfaction of passengers.

This research's evaluation process was designed to evaluate the hypothesis that "Vehicle Routing Problem can be solved by Multi Agent Systems". According to the results of evaluation, it found 80% of solutions, which are generated by Multi Agent System, give efficient solutions than manually generated solutions. Furthermore, most of the solutions were having more than 5% of efficiency improvement which is considerable amount of cost saving. Based on these results it is possible to arrive at the conclusion that the hypothesis ("Vehicle routing problem can be solved by Multi Agent Systems") is correct.

8.3 Limitations and Further Work

When running these testing, the number of vehicle was added as a manual input to maximize the quality of the results. When the number of vehicles is higher than the required level, it tends to generate shuttle plans with a slightly higher total route cost. To avoid this it is necessary to develop a negotiation process between shuttles to identify the number of vehicles which gives best results by themselves.

When analyzing the results of the evaluation, two datasets were found which do not agree with the hypothesis. When further analyzing these two results, it was found that sometimes passenger groups were not flexible enough when they negotiate with vehicles to do shuttle allocation. For example, when there are two groups with 6 or 7 passengers, they needed separate vehicles and did not agree to break the groups and join extra passengers to another group. This matter should be improved in future to get more accurate results. In the current solution these are the improvements that have been identified through this process.

8.4 Summary

In this chapter we have discussed the conclusions that we can finally derive from the research. All the aspects of the problem, technology, design, implementation, evaluation were discussed. Based on all the facts, a final conclusion was stated showing how the Vehicle Routine Problem can be efficiently solved by the Multi Agent System. If there would be further studying and research to solve the problem encountered as stated in further work, the system would showcase higher performance and will function as a perfect solution to all the parties that are involved.

References

[1] G. Dantzig and J. Ramser: "The Truck Dispatching Problem": Management Science Volume. 6: 1959

[2] G. Clarke and J. R. Wright: "Scheduling of vehicles from a central depot to a number of delivery points": Operations Research Vol. 12: 1964

[3] Suresh Nanda Kumar, Ramasamy Panneerselvam: "A Survey on the Vehicle Routing Problem and Its Variants": Intelligent Information Management: 2012

[4] J. K. Lenstra and A. H. G. Rinnooy Kan: "Complexity of Vehicle and Scheduling Problems": Networks, Vol. 11: 1981

[5] J. Homberger and H. Gehring: "A Two-Phase Hybrid Meta-Heuristic for the Vehicle Routing Problem with Time Windows": European Journal of Operational Research: Vol. 162: 2005

[6] F. J. Bard, G. Kontoravdis and G. Yu: "A Branch-and-Cut Procedure for the Vehicle Routing Problem with Time Windows": Transportation Science, Vol. 36: 2002

[7] G. Gutiérrez-Jarpa, G. Desaulniers, G. Laporte and V. Marianov: "A Branch-and-Price Algorithm for the Vehicle Routing Problem with Deliveries, Selective Pickups and Time Windows": European Journal of Operational Research Vol. 206, No. 12: 2010

[8] G. B. Alvarenga, G. R. Mateus and G. de Tomi: "A Genetic and Set Partitioning Two-Phase Approach for the Vehicle Routing Problem with Time Windows": Computers and Operations Research, Vol. 34, No. 6: 2007

[9] P. Shaw: "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems": University of Strathclyde, Glasgow: 1998

[10] M. Gandreau, F. Guertin, J.-Y. Potvin and R. Seguin: "Neighborhood Search Heuristics for a Dynamic Vehicle Dispatching Problem with Pick-Ups and Deliveries": Transportation Research Part E, Logistics and Transportation Review, sportation ReVol. 14: 2006

[11] Geir Hasle, Knut-Andreas Lie, Ewald Quak; "Geometric modelling, numerical simulation, and optimization applied mathematics": 2007

[12] George Rzevski, Jonathan Himoff and Petr Skobelev: "MAGENTA Technology Multi-Agent Logistics i-Scheduler for Road Transportation"

[13] Petr Skobelev: "Bio-Inspired Multi-Agent Technology for Industrial Applications"

[14] George Rzevski, Petr Skobele: "Emergent Intelligence in Large Scale Multi-Agent Systems: International journal of education and information technologies"

[15] Olli Bräysy, Michel Gendreau: "Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms": Transportation Science Volume 39 Issue 1, February 2005

[16] Gilbert Laporte: "The Vehicle Routing Problem: An overview of exact and approximate algorithms"

[17] "Agent Technology: Computing as Interaction. A Roadmap for Agent Based Computing" : http://www.agentlink.org/roadmap/index.html

[18] Yung-yu Tseng, Wen Long Yue and Michael A P Taylor: "The role of transportation in logistics chain": Proceedings of the Eastern Asia Society for Transportation Studies, Vol. 5, pp. 1657 - 1672, 2005

Appendix A

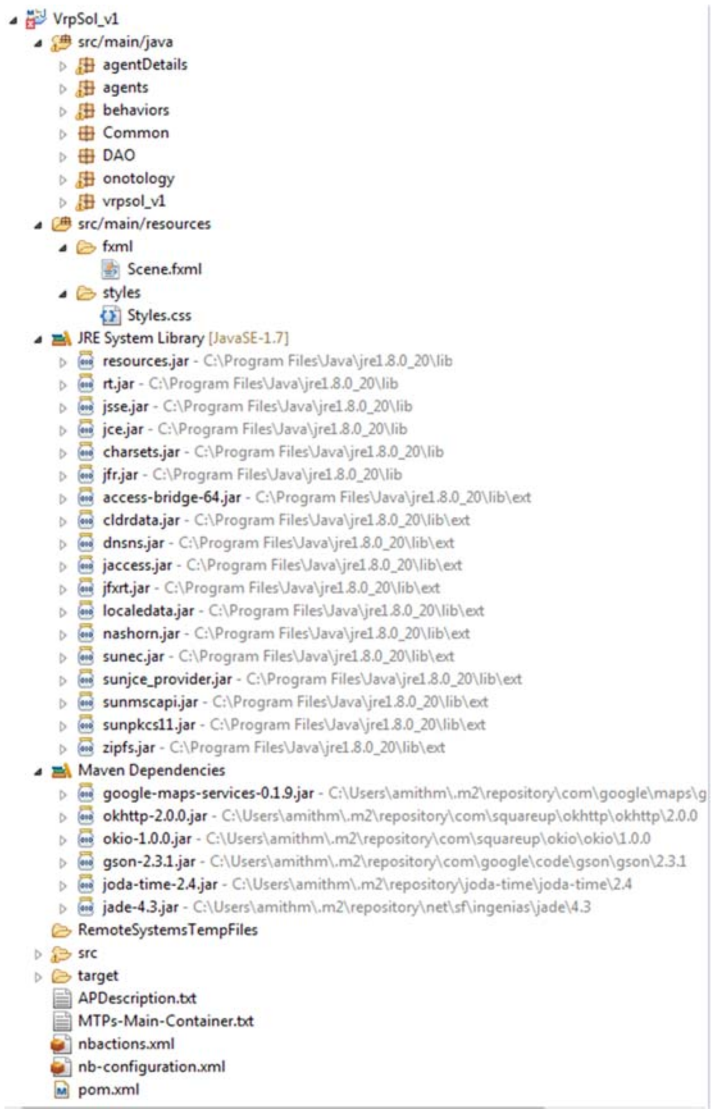
Source code of the system

A.1 Introduction

This section will include source code which was used to develop the proposed system.

A.2 Sample ource code of the system

<u>Structure of the project</u>



```
VrpSol_v1
  src/main/java
    agentDetails
    agents
    behaviors
    Common
    DAO
    onotology
    vrpsol_v1
  src/main/resources
    fxml
      Scene.fxml
    styles
      Styles.css
  JRE System Library [JavaSE-1.7]
    resources.jar - C:\Program Files\Java\jre1.8.0_20\lib
    rt.jar - C:\Program Files\Java\jre1.8.0_20\lib
    jsse.jar - C:\Program Files\Java\jre1.8.0_20\lib
    jce.jar - C:\Program Files\Java\jre1.8.0_20\lib
    charsets.jar - C:\Program Files\Java\jre1.8.0_20\lib
    jfr.jar - C:\Program Files\Java\jre1.8.0_20\lib
    access-bridge-64.jar - C:\Program Files\Java\jre1.8.0_20\lib\ext
    cldrdata.jar - C:\Program Files\Java\jre1.8.0_20\lib\ext
    dnsns.jar - C:\Program Files\Java\jre1.8.0_20\lib\ext
    jaccess.jar - C:\Program Files\Java\jre1.8.0_20\lib\ext
    jfxrt.jar - C:\Program Files\Java\jre1.8.0_20\lib\ext
    localedata.jar - C:\Program Files\Java\jre1.8.0_20\lib\ext
    nashorn.jar - C:\Program Files\Java\jre1.8.0_20\lib\ext
    sunec.jar - C:\Program Files\Java\jre1.8.0_20\lib\ext
    sunjce_provider.jar - C:\Program Files\Java\jre1.8.0_20\lib\ext
    sunmscapi.jar - C:\Program Files\Java\jre1.8.0_20\lib\ext
    sunpkcs11.jar - C:\Program Files\Java\jre1.8.0_20\lib\ext
    zipfs.jar - C:\Program Files\Java\jre1.8.0_20\lib\ext
  Maven Dependencies
    google-maps-services-0.1.9.jar - C:\Users\amithm\.m2\repository\com\google\maps\g
    okhttp-2.0.0.jar - C:\Users\amithm\.m2\repository\com\squareup\okhttp\okhttp\2.0.0
    okio-1.0.0.jar - C:\Users\amithm\.m2\repository\com\squareup\okio\okio\1.0.0
    gson-2.3.1.jar - C:\Users\amithm\.m2\repository\com\google\code\gson\gson\2.3.1
    joda-time-2.4.jar - C:\Users\amithm\.m2\repository\joda-time\joda-time\2.4
    jade-4.3.jar - C:\Users\amithm\.m2\repository\net\sf\ingenias\jade\4.3
  RemoteSystemsTempFiles
  src
  target
  APDescription.txt
  MTPs-Main-Container.txt
  nbactions.xml
  nb-configuration.xml
  pom.xml
```

Figure A.1: Structure of the project

JAVA class structure

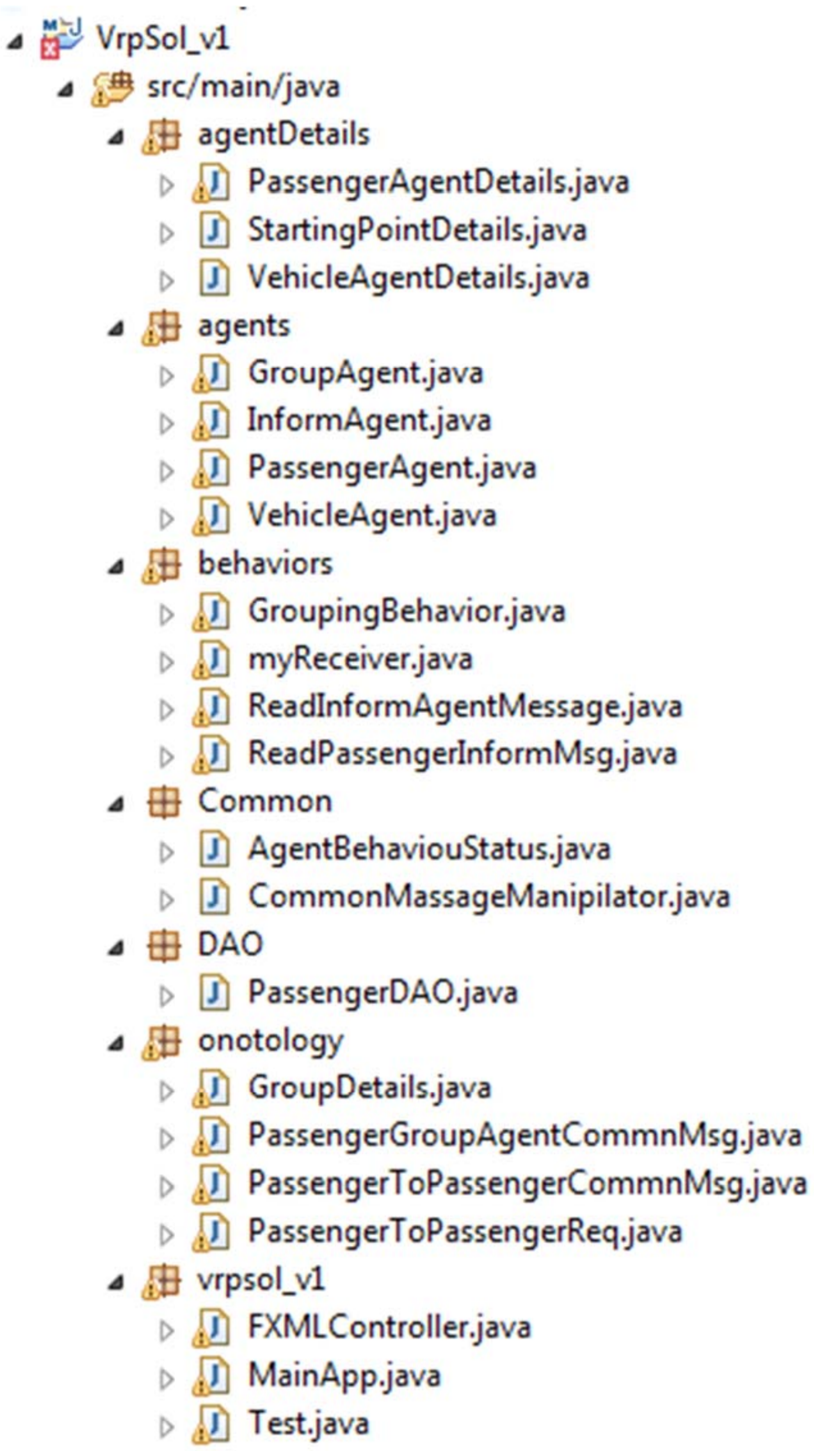

VrpSol_v1
src/main/java
agentDetails
PassengerAgentDetails.java
StartingPointDetails.java
VehicleAgentDetails.java
agents
GroupAgent.java
InformAgent.java
PassengerAgent.java
VehicleAgent.java
behaviors
GroupingBehavior.java
myReceiver.java
ReadInformAgentMessage.java
ReadPassengerInformMsg.java
Common
AgentBehaviouStatus.java
CommonMassageManipilator.java
DAO
PassengerDAO.java
onotology
GroupDetails.java
PassengerGroupAgentCommnMsg.java
PassengerToPassengerCommnMsg.java
PassengerToPassengerReq.java
vrpsol_v1
FXMLController.java
MainApp.java
Test.java

Figure A.2: Class Structure

MainApp.java

This java class works as main class of the project. This is responsible to start the application and load GUI to screen.

```
package vrpsol_v1;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;


public class MainApp extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("/fxml/Scene.fxml"));

        Scene scene = new Scene(root);
        scene.getStylesheets().add("/styles/Styles.css");
        stage.setTitle("JavaFX and Maven");
        stage.setScene(scene);
        stage.show();

    }

    public static void main(String[] args) {
        launch(args);
    }

}
```

FXMLController.java

This Java class is controlling all user interactions with the system.

```
package vrpsol_v1;

import Common.AgentBehaviouStatus;
import Common.CommonMassageManipilator;
import DAO.PassengerDAO;
import agentDetails.PassengerAgentDetails;
import agentDetails.StartingPointDetails;
import static com.sun.corba.se.impl.util.Utility.printStackTrace;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
import jade.wrapper.StaleProxyException;
import java.io.File;
import java.net.URL;
import java.util.List;
import java.util.ResourceBundle;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;

public class FXMLController implements Initializable {

    private Label lable1;
    jade.core.Runtime rt = jade.core.Runtime.instance();
    Profile profile = new ProfileImpl(null, 1099, null);
    AgentContainer mainContainer = rt.createMainContainer(profile);

    @FXML
    private Label lblNoOfVeh;
    @FXML
    private Label lblPassengers;
    @FXML
    private TextField txtNoOfVeh;
    @FXML
    private TextArea txtPassengers;
    @FXML
    private Button btnNoOfVeh;
    @FXML
    private Button btnStartNormalMode;

    @FXML
    private Button btnAdminMode;

    @FXML
    private Button btnNormalMode;

    @FXML
    private Button btnAddPassengers;
    @FXML
    private Button btnGroupPassengers;
    @FXML
    public WebView webView;
    public static WebView staticWebview;
    public static WebEngine staticWebEngine;

    @FXML
    public void startAdminMode(ActionEvent event) {
        try {
            WebEngine engine = webView.getEngine();
            File mapFile = new File("E:/html/Map.html");
            engine.load(mapFile.toURI().toURL().toString());
            rt.setCloseVM(true);
            staticWebview = webView;
            staticWebEngine = engine;
            AgentController agent = mainContainer.createNewAgent("rma", 
"jade.tools.rma.rma", new Object[0]);
            agent.start();
            lblNoOfVeh.setDisable(false);
            txtNoOfVeh.setDisable(false);
            btnNoOfVeh.setDisable(false);
            lblNoOfVeh.setVisible(true);
            txtNoOfVeh.setVisible(true);
            btnNoOfVeh.setVisible(true);
            btnNormalMode.setDisable(true);
            btnAdminMode.setDisable(true);
        } catch (Exception e) {
            printStackTrace();
        }
    }

    @FXML
    public void selectNormalMode(ActionEvent event) {
        try {
            WebEngine engine = webView.getEngine();
            File mapFile = new File("E:/html/Map.html");
            engine.load(mapFile.toURI().toURL().toString());
            rt.setCloseVM(true);
            staticWebview = webView;
            staticWebEngine = engine;
            AgentController agent = mainContainer.createNewAgent("rma", 
"jade.tools.rma.rma", new Object[0]);
            agent.start();
            addVeihcles(10);
            btnStartNormalMode.setVisible(true);
            //btnNormalMode.setDisable(true);
            //btnAdminMode.setDisable(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @FXML
    public void startNormalMode(ActionEvent event) {
        try {
            List<PassengerAgentDetails>  passengers=
PassengerDAO.getRequestedPassengerList();
            addPassengers(passengers);
            AgentController informAgent = mainContainer.createNewAgent("informAgent",
"agents.InformAgent", new Object[0]);
            informAgent.start();
            AgentController groupAgent = mainContainer.createNewAgent("GroupAgent",
"agents.GroupAgent", new Object[0]);
            groupAgent.start();
        } catch (Exception e) {
            printStackTrace();
        }
    }
    @FXML
    public void addNoOfVehicles(ActionEvent event) {
        String strNoOfVehicles = txtNoOfVeh.getText();
        Integer intNoOfVehicles = 0;
        if (strNoOfVehicles != null && strNoOfVehicles.length() != 0) {
            intNoOfVehicles = Integer.parseInt(strNoOfVehicles);
            addVeihcles(intNoOfVehicles);
            txtNoOfVeh.setDisable(true);
            btnNoOfVeh.setDisable(true);
            lblPassengers.setDisable(false);
            txtPassengers.setDisable(false);
            btnAddPassengers.setDisable(false);
            lblPassengers.setVisible(true);
            txtPassengers.setVisible(true);
            btnAddPassengers.setVisible(true);
        }
    }

    @FXML
    public void addPassengers(ActionEvent event) throws StaleProxyException {
        String strPassengers = txtPassengers.getText();
        CommonMassageManipilator msgManipuManipilator = new
CommonMassageManipilator();
        List<PassengerAgentDetails> passengers =
msgManipuManipilator.getPassengerList(strPassengers);
        Integer noOfPassengers = passengers.size();
        if (passengers != null && passengers.size() != 0) {
            addPassengers(passengers);
            lblPassengers.setDisable(true);
            txtPassengers.setDisable(true);
            btnAddPassengers.setDisable(true);
            btnGroupPassengers.setDisable(false);
            AgentController informAgent = mainContainer.createNewAgent("informAgent",
"agents.InformAgent", new Object[0]);
            informAgent.start();
            AgentController groupAgent = mainContainer.createNewAgent("GroupAgent",
"agents.GroupAgent", new Object[0]);
            groupAgent.start();
        }
    }
    @FXML
    public void groupPassengers(ActionEvent event) {
        try {
            AgentBehaviouStatus.informToStartGroupPassengersDone = false;

        } catch (Exception ex) {
            Logger.getLogger(FXMLController.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }

    public boolean addVeihcles(int noOfVehicles) {
        try {
            for (int i = 1; i <= noOfVehicles; i++) {
                AgentController vehicleAgent;
                try {
                    vehicleAgent = mainContainer.createNewAgent("Vehicle" + i,
"agents.VehicleAgent", new Object[0]);
                    vehicleAgent.start();
                } catch (StaleProxyException ex) {

Logger.getLogger(FXMLController.class.getName()).log(Level.SEVERE, null, ex);
                }

            }
        } catch (Exception ex) {
            ex.printStackTrace();
            return false;
        }
        return true;
    }

    public boolean addPassengers(List<PassengerAgentDetails> passengers) {
        try {
            Integer noOfPassengers = passengers.size();
            if (passengers != null && passengers.size() != 0) {
                for (int i = 0; i < noOfPassengers; i++) {
                    WebEngine engine = webView.getEngine();
                    engine.executeScript("addMarker('" +
passengers.get(i).getAgentName() + "'," + passengers.get(i).getLatitude() + "," +
passengers.get(i).getLongitude() + ")");
                    PassengerAgentDetails passengerAgentDetails = new
PassengerAgentDetails(passengers.get(i).getAgentName(),
passengers.get(i).getLatitude(), passengers.get(i).getLongitude());
                    AgentController passengerAgent;
                    try {
                        passengerAgent =
mainContainer.createNewAgent(passengers.get(i).getAgentName(),
"agents.PassengerAgent", new Object[]{passengerAgentDetails});
                        passengerAgent.start();
                    } catch (StaleProxyException ex) {
```

```
Logger.getLogger(FXMLController.class.getName()).log(Level.SEVERE, null, ex);
                    }

                }
            }
        } catch (Exception ex) {
            ex.printStackTrace();
            return false;
        }
        return true;
    }

    public static void markGroupMember(PassengerAgentDetails passenger, String
iconColor){
        if(iconColor==null){
            iconColor = "FE7569";
        }
        staticWebEngine.executeScript("addMarkerWithColor('" +
passenger.getAgentName() + "'," + passenger.getLatitude() + "," +
passenger.getLongitude() + ",'"+iconColor+"')");
    }

    public static void deleteMarkers(){
        staticWebEngine.executeScript("deleteMarkers()");
    }

    public static void displayRoute(List<PassengerAgentDetails> assignedPassengers){
        String routeDeitals =
"['MIT',"+StartingPointDetails.latitude+","+StartingPointDetails.longitude+"]";
        for (PassengerAgentDetails assignedPassenger : assignedPassengers) {

routeDeitals=routeDeitals+",['"+assignedPassenger.getAgentName()+"',"+assignedPasseng
er.getLatitude()+","+assignedPassenger.getLongitude()+"]";
        }
        System.out.println("calculateAndDisplayRoute(["+routeDeitals+"])");

staticWebEngine.executeScript("calculateAndDisplayRoute(["+routeDeitals+"])");

    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

}
```

<u>PassengerAgent.java</u>

package agents;

```
import agentDetails.PassengerAgentDetails;
import agentDetails.StartingPointDetails;
import behaviors.ReadInformAgentMessage;
import com.google.maps.DirectionsApi;
import com.google.maps.GeoApiContext;
import com.google.maps.model.DirectionsRoute;
import com.google.maps.model.LatLng;
import com.google.maps.model.TravelMode;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;
import onotology.GroupDetails;

import onotology.PassengerToPassengerReq;

import org.joda.time.DateTime;


/**

 *

 * @author amithm

 */

public class PassengerAgent extends Agent {


	private double expectedMinimumTravalTime;

	private double expectedMaximumTravalTime;

	private PassengerAgentDetails passengerAgentDetails;

	private Set<PassengerAgentDetails> nearByPassengers = new
HashSet<PassengerAgentDetails>();

	private GroupDetails groupDetails = new GroupDetails();

	private boolean creationIsFinished = false;

	private boolean negotiatingPassenger = false;

	private boolean inaGroup = false;

	GeoApiContext context = new GeoApiContext()

			.setApiKey("AIzaSyAQeLDVDVACYKFX2oMdKkB0BXPneIGYkD4");


	protected void setup() {

		System.out.println("Passenger agent " + getLocalName() + " started");

		addBehaviour(new createRequiredInfo(this));
```

```
        addBehaviour(new ReadInformAgentMessage(this));

}

// ......... Start createRequiredInfoBehaviour ...........
class createRequiredInfo extends SimpleBehaviour {

        SimpleDateFormat time_formatter = new SimpleDateFormat(
                        "yyyy-MM-dd_HH:mm:ss.SSS");
        Agent thisAgent;

        public createRequiredInfo(Agent a) {
                super(a);
                thisAgent = a;
        }

        protected void setup() {
                ServiceDescription sd = new ServiceDescription();
                sd.setType("passenger");
                sd.setName(getLocalName());
                register(sd);
        }

        void register(ServiceDescription sd) {
```

```
		DFAgentDescription dfd = new DFAgentDescription();
		dfd.setName(getAID());
		dfd.addServices(sd);

		try {
			DFService.register(thisAgent, dfd);
		} catch (FIPAException fe) {
			fe.printStackTrace();
		}
	}

	public void action() {
		Object[] args = getArguments();
		passengerAgentDetails = (PassengerAgentDetails) args[0];

		try {
			LatLng passengerLatLng = new LatLng(
					passengerAgentDetails.getLatitude(),
					passengerAgentDetails.getLongitude());
			DirectionsRoute[] routes = DirectionsApi.newRequest(context)
					.mode(TravelMode.DRIVING).departureTime(new
DateTime())
					.origin(StartingPointDetails.latLan)
					.destination(passengerLatLng).await();
```

```
                                expectedMinimumTravalTime =
routes[0].legs[0].duration.inSeconds;
                                expectedMaximumTravalTime = expectedMinimumTravalTime
                                            + expectedMinimumTravalTime / 5;
                                passengerAgentDetails

        .setExpectedMinimumTravalTime(expectedMinimumTravalTime);
                                passengerAgentDetails

        .setExpectedMaximumTravalTime(expectedMaximumTravalTime);
                                System.out.println(time_formatter.format(System
                                            .currentTimeMillis())
                                            + " : "
                                            + passengerAgentDetails.getAgentName()
                                            + "'s maxsimum expected travel time is "
                                            + expectedMaximumTravalTime);
                                creationIsFinished = true;

                                DFAgentDescription dfd = new DFAgentDescription();
                                dfd.setName(getAID());
                                ServiceDescription sd = new ServiceDescription();
                                sd.setType("passenger");
                                sd.setName(getLocalName());
                                dfd.addServices(sd);
                                DFService.register(thisAgent, dfd);
```

```
                } catch (Exception ex) {
                        Logger.getLogger(PassengerAgent.class.getName()).log(
                                        Level.SEVERE, null, ex);
                }
        }

        public boolean done() {
                return creationIsFinished;
        }
}

// ......... End createRequiredInfoBehaviour .............

public PassengerAgentDetails getPassengerAgentDetails() {
        return passengerAgentDetails;
}

public double getExpectedMinimumTravalTime() {
        return expectedMinimumTravalTime;
}

public Set<PassengerAgentDetails> getNearByPassengers() {
        return nearByPassengers;
}
```

```
public void setNearByPassengers(Set<PassengerAgentDetails> nearByPassengers) {
	this.nearByPassengers = nearByPassengers;
}

public GeoApiContext getContext() {
	return context;
}

public void setContext(GeoApiContext context) {
	this.context = context;
}

public double getExpectedMaximumTravalTime() {
	return expectedMaximumTravalTime;
}

public void setExpectedMaximumTravalTime(double expectedMaximumTravalTime) {
	this.expectedMaximumTravalTime = expectedMaximumTravalTime;
}

public GroupDetails getGroupDetails() {
	return groupDetails;
}
```

```
public void setGroupDetails(GroupDetails groupDetails) {
        this.groupDetails = groupDetails;
}

public boolean isCreationIsFinished() {
        return creationIsFinished;
}

public void setCreationIsFinished(boolean creationIsFinished) {
        this.creationIsFinished = creationIsFinished;
}

public boolean isNegotiatingPassenger() {
        return negotiatingPassenger;
}

public void setNegotiatingPassenger(boolean negotiatingPassenger) {
        this.negotiatingPassenger = negotiatingPassenger;
}

public boolean isInaGroup() {
        return inaGroup;
}
```

```
        public void setInaGroup(boolean inaGroup) {
                this.inaGroup = inaGroup;
        }
}
```

VehicleAgent.java

```
package agents;

import agentDetails.PassengerAgentDetails;
import behaviors.VehicleCommunicatingBehavior;
import jade.core.Agent;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import java.util.HashSet;
import java.util.Set;

/**
 *
 * @author amithm
 */
public class VehicleAgent extends Agent {
    private int noOfSeats = 10;

    private int seatsAvailable;

    private PassengerAgentDetails firstPassenger;

    private PassengerAgentDetails lastPassenger;

    private Set<PassengerAgentDetails> assignedPassengers = new
HashSet<PassengerAgentDetails>();


    protected void setup()

    {

        System.out.println("Vehicle agent "+ getLocalName()+" started");

        ServiceDescription sd = new ServiceDescription();

        sd.setType("vehicle");

        sd.setName(getLocalName());

        register(sd);

        addBehaviour(new VehicleCommunicatingBehavior(this));

    }

    void register(ServiceDescription sd) {

            DFAgentDescription dfd = new DFAgentDescription();

            dfd.setName(getAID());

            dfd.addServices(sd);

            try {

                DFService.register(this, dfd);

            } catch (FIPAException fe) {

                fe.printStackTrace();
        }
    }

    public int getNoOfSeats() {
        return noOfSeats;
    }

    public void setNoOfSeats(int noOfSeats) {
        this.noOfSeats = noOfSeats;
    }

    public int getSeatsAvailable() {
        return seatsAvailable;
    }

    public void setSeatsAvailable(int seatsAvailable) {
        this.seatsAvailable = seatsAvailable;
    }

    public Set<PassengerAgentDetails> getAssignedPassengers() {
        return assignedPassengers;
    }

    public void setAssignedPassengers(Set<PassengerAgentDetails> assignedPassengers) {
        this.assignedPassengers = assignedPassengers;
    }

    public PassengerAgentDetails getFirstPassenger() {
        return firstPassenger;
    }

    public void setFirstPassenger(PassengerAgentDetails firstPassenger) {
        this.firstPassenger = firstPassenger;
    }

    public PassengerAgentDetails getLastPassenger() {
        return lastPassenger;
    }

    public void setLastPassenger(PassengerAgentDetails lastPassenger) {
        this.lastPassenger = lastPassenger;
    }

}
```

GroupAgent.java

```
package agents;
```

```
import agentDetails.PassengerAgentDetails;

import onotology.GroupDetails;

import behaviors.GroupingBehavior;

import jade.core.Agent;

import jade.domain.DFService;

import jade.domain.FIPAAgentManagement.DFAgentDescription;

import jade.domain.FIPAAgentManagement.ServiceDescription;

import java.io.IOException;

import java.util.HashMap;

import java.util.HashSet;

import java.util.List;

import java.util.ArrayList;

import java.util.logging.Level;

import java.util.logging.Logger;


/**

 *

 * @author amithm

 */

public class GroupAgent extends Agent {



   private List<GroupDetails> groups = new ArrayList<GroupDetails>();

   private HashMap<PassengerAgentDetails, HashSet<PassengerAgentDetails>>
nearbyAgentsOfPassengers = new HashMap<PassengerAgentDetails,
HashSet<PassengerAgentDetails>>();
```

```
protected void setup() {
    ServiceDescription sd = new ServiceDescription();
    sd.setType("GroupAgent");
    sd.setName(getLocalName());
    register(sd);
    try {
        addBehaviour(new GroupingBehavior(this));
    } catch (IOException ex) {
        Logger.getLogger(GroupAgent.class.getName()).log(Level.SEVERE, null, ex);
    }
}

void register(ServiceDescription sd) {
    try {
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());
        dfd.addServices(sd);
        DFService.register(this, dfd);
    } catch (Exception fe) {
        fe.printStackTrace();
    }
}
```

```
   public HashMap<PassengerAgentDetails, HashSet<PassengerAgentDetails>>
getNearbyAgentsOfPassengers() {
        return nearbyAgentsOfPassengers;
   }

   public void setNearbyAgentsOfPassengers(HashMap<PassengerAgentDetails,
HashSet<PassengerAgentDetails>> nearbyAgentsOfPassengers) {
        this.nearbyAgentsOfPassengers = nearbyAgentsOfPassengers;
   }

   public List<GroupDetails> getGroups() {
        return groups;
   }

   public void setGroups(List<GroupDetails> groups) {
        this.groups = groups;
   }

}
```

<u>InformAgent.java</u>

```
package agents;

import Common.AgentBehaviouStatus;
import agentDetails.PassengerAgentDetails;
import agentDetails.StartingPointDetails;
import behaviors.AssignedDetailsViewBehavior;
import com.google.maps.DirectionsApi;
import com.google.maps.GeoApiContext;
import com.google.maps.model.DirectionsRoute;
import com.google.maps.model.LatLng;
import com.google.maps.model.TravelMode;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.joda.time.DateTime;

/**
 *
 * @author amithm
 */
public class InformAgent extends Agent {


    private boolean informedToGroup = false;
    private int noOfPassengerAgents = 0;


    protected void setup() {
        System.out.println("Inform agent " + getLocalName() + " started");
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());
        ServiceDescription sd = new ServiceDescription();
        sd.setType("InformAgent");
        sd.setName(getLocalName());
        dfd.addServices(sd);
        try {
            DFService.register(this, dfd);
        } catch (FIPAException ex) {
            Logger.getLogger(InformAgent.class.getName()).log(Level.SEVERE, null, ex);
        }
        addBehaviour(new InformToStartGroupPassengers(this));
        try {
            addBehaviour(new AssignedDetailsViewBehavior(this));
        } catch (IOException ex) {
            Logger.getLogger(InformAgent.class.getName()).log(Level.SEVERE, null, ex);
        } catch (FIPAException ex) {
            Logger.getLogger(InformAgent.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    class InformToStartGroupPassengers extends SimpleBehaviour {

        SimpleDateFormat time_formatter = new SimpleDateFormat("yyyy-MM-
dd_HH:mm:ss.SSS");
        Agent thisAgent;
        int n = 0;

        public InformToStartGroupPassengers(Agent a) {
            super(a);
            thisAgent = a;
        }

        public void action() {
            try {

                if (n == 1) {
                    System.out.println("Grouping passengers started.");
                    DFAgentDescription dfd = new DFAgentDescription();
                    ServiceDescription sd = new ServiceDescription();
                    sd.setType("passenger");
                    dfd.addServices(sd);

                    DFAgentDescription[] result = DFService.search(thisAgent, dfd);

                    result = DFService.search(thisAgent, dfd);

                    noOfPassengerAgents = result.length;

                    System.out.println("Sending grouping message to " + result.length + " of
    Passengers.");

                    for (int i = 0; i < result.length; i++) {

                        System.out.println("Sending message to Passenger " + i + " by Inform Agent");

                        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);

                        msg.setContent("Start_Grouping");

                        AID dest = result[i].getName();

                        msg.addReceiver(dest);

                        send(msg);

                    }

                }

                block(10000);

                n++;

            } catch (Exception ex) {

                ex.printStackTrace();

            }

        }


        public boolean done() {

            System.out.println("n is " + n);

            return n >= 2;
        }
    }

    public int getNoOfPassengerAgents() {
        return noOfPassengerAgents;
    }

    public void setNoOfPassengerAgents(int noOfPassengerAgents) {
        this.noOfPassengerAgents = noOfPassengerAgents;
    }
}
```

PassengerAgentDetails.java

```
package agentDetails;

import java.io.Serializable;
import java.util.Comparator;
import java.util.Objects;

/**
 *
 * @author amithm
 */
public class PassengerAgentDetails implements Serializable{
    private String AgentName;
    private double latitude;
    private double longitude;
    private double expectedMinimumTravalTime;
    private double expectedMaximumTravalTime;
    private double acceptedMinTravalTime;
    private double acceptedMaxTravalTime;

    public PassengerAgentDetails() {
    }

    public PassengerAgentDetails(String AgentName, double latitude, double longitude) {
        this.AgentName = AgentName;
        this.latitude = latitude;
        this.longitude = longitude;
    }

    public PassengerAgentDetails(String AgentName, double latitude, double longitude, double
expectedMinimumTravalTime, double expectedMaximumTravalTime, double
acceptedMinTravalTime, double acceptedMaxTravalTime) {
        this.AgentName = AgentName;
        this.latitude = latitude;
        this.longitude = longitude;
        this.expectedMinimumTravalTime = expectedMinimumTravalTime;
        this.expectedMaximumTravalTime = expectedMaximumTravalTime;
    this.acceptedMinTravalTime = acceptedMinTravalTime;
    this.acceptedMaxTravalTime = acceptedMaxTravalTime;
}

public String getAgentName() {
    return AgentName;
}

public void setAgentName(String AgentName) {
    this.AgentName = AgentName;
}

public double getLatitude() {
    return latitude;
}

public void setLatitude(double latitude) {
    this.latitude = latitude;
}

public double getLongitude() {
    return longitude;
}
```

```
public void setLongitude(double longitude) {
    this.longitude = longitude;
}

public double getExpectedMinimumTravalTime() {
    return expectedMinimumTravalTime;
}

public void setExpectedMinimumTravalTime(double expectedMinimumTravalTime) {
    this.expectedMinimumTravalTime = expectedMinimumTravalTime;
}

public double getExpectedMaximumTravalTime() {
    return expectedMaximumTravalTime;
}

public void setExpectedMaximumTravalTime(double expectedMaximumTravalTime) {
    this.expectedMaximumTravalTime = expectedMaximumTravalTime;
}

public double getAcceptedMinTravalTime() {
    return acceptedMinTravalTime;
}
```

```
public void setAcceptedMinTravalTime(double acceptedMinTravalTime) {
    this.acceptedMinTravalTime = acceptedMinTravalTime;
}

public double getAcceptedMaxTravalTime() {
    return acceptedMaxTravalTime;
}

public void setAcceptedMaxTravalTime(double acceptedMaxTravalTime) {
    this.acceptedMaxTravalTime = acceptedMaxTravalTime;
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 47 * hash + Objects.hashCode(this.AgentName);
    return hash;
}

@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
```

```
        if (!(obj instanceof PassengerAgentDetails)) {
            return false;
        }
        PassengerAgentDetails other = (PassengerAgentDetails) obj;
        if (this.getAgentName().equals(other.getAgentName()) ) {
            return true;
        }

        else{
            return false;
        }
    }

}
```