

# Multi Agent System for Ride Sharing and Carpooling



University of Moratuwa, Sri Lanka.  
P.K.H.A. Sirisena  
Electronic Theses & Dissertations  
129109H  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

Faculty of Information Technology

University of Moratuwa

August 2014

# Multi Agent System for Ride Sharing and Carpooling



University of Moratuwa, Sri Lanka.  
P.K.H.A. Sirisena  
Electronic Theses & Dissertations  
129109H  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

Dissertation submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka for the partial fulfillment of the requirements of the Degree of MSc in Artificial Intelligence

February 2015

## Declaration

I declare that this dissertation 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 dissertation, if accepted, to be made available for photocopying and for interlibrary loans, and for the title and summary to be made available to outside organizations.

HariniSirisena  
Name of Student



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

Signature of Student

Date:

Supervised by

Prof. Asoka Karunananda

Name of Supervisor

Signature of Supervisor

Date:

## Acknowledgement

I am grateful for all the support I received from my lecturers, friends and family to complete this research thesis. I extend my gratitude to my lecturers from my postgraduate course who have assisted me in my studies and my research work. Special thanks to Prof. Asoka Karunanada who supervised this project and provided valuable insight and advice. Last but not least I would like to thank all my friends and family who have supported me throughout my studies.



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)



# Abstract

This thesis describes the research done to implement a ridesharing solution in Sri Lanka using a multi-agent system based approach. Particularly this research focuses on implementing a carpooling/ridesharing solution using real geospatial data extracted from a geographic information system. Emphasis is placed on solving the complex problem of user route matching based on journey start, end locations and route overlap. Carpooling is the sharing of car journeys so that more than one person travels in a car. By having more people using one vehicle, carpooling reduces each person's travel costs such as fuel costs, tolls, and the stress of driving. Carpooling is seen as a more environmentally friendly and sustainable way to travel, since sharing journeys reduces carbon emissions, traffic congestion on the roads, and the need for parking spaces. Carpooling/ridesharing however has many challenges such as socio-cultural challenges, which need to be looked into when implementing a solution.

A multi agent based solution is proposed due to the complex and dynamic nature of the problem. For this system, users are simulated using the simulation capabilities of a multi agent system. Route data for the simulated users are generated using a list of addresses in the Western Province of Sri Lanka. The list of addresses are then converted to geo location data and randomly paired using a randomized pairing algorithm to generate routes for the users. Agents generated with the simulated routes are allowed to interact with one another to group together and form ride shares or carpools. A Route Match Agent is implemented with a custom route match algorithm in order to find the best pairings for ride shares and carpools. Carpools are identified as an extension of rideshares, established between multiple users who have overlapping journeys with approximately the same route distances. The system also includes a social network module where connections between users are mapped as first and second degree connections. The available rideshares/carpools for user are then ranked based on the social connections. Running tests against the system showed that it is effective in finding the optimal carpools for users. The simulation also showed that for successful carpools to be established, a large user pool from the same area is required.

# Contents

<b>Chapter 1 Introduction.....</b>	<b>1</b>
1.1 Prolegomenon .....	1
1.3 Problem definition .....	1
1.4 Aim and Objectives.....	3
1.5 Outline.....	3
1.6 Summary .....	4
<b>Chapter 2 Analysis of carpooling/ridesharing solutions .....</b>	<b>5</b>
2.1 Introduction.....	5
2.2 Research into the challenges of carpooling and ridesharing.....	5
2.3 Solving the carpooling problem using multi agent systems .....	6
2.4 Other solutions to the carpooling problem.....	10
2.5 Summary .....	12
<b>Chapter 3 Multi Agent Systems and their applications.....</b>	<b>13</b>
3.1 Introduction.....	13
3.2 Agent Definition .....	13
3.3 Multi agent system implementation.....	15
3.4 Multi agent system architecture .....	13
3.5 Communication, cordination and negotiation.....	13
3.6 JADE.....	20
3.7 MAS in Traffic and Transportation .....	20
3.8 Summary .....	21
<b>Chapter 4 Solving complexity using a multi agent system.....</b>	<b>22</b>
4.1 Introduction.....	22
4.2 Hypothesis.....	22
4.3 System requirements and features .....	22
4.4 Solution overview .....	23
4.5 Summary .....	24



University of Moratuwa, Sri Lanka.

Electronic Theses & Dissertations

[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

<b>Chapter 5 Design.....</b>	<b>25</b>
5.1 Introduction.....	25
5.2 System layered architecture .....	25
5.3 Interface Layer .....	26
5.4 Agent Layer .....	26
5.5 Ontology Layer .....	27
5.6 System activity diagrams .....	28
5.7 Summary .....	32
<b>Chapter 6 Implementation .....</b>	<b>33</b>
6.1 Introduction.....	33
6.2 Interface module implementation .....	33
6.3 Agent layer implementation.....	33
6.4 JADE agent behaviors.....	33
6.5 System agents.....	35
6.6 Route Match Agent Algorithm .....	38
6.7 Ontology layer implementation .....	40
6.8 Database layer implementation.....	40
6.9 Social network module .....	40
6.10 Maps data Layer.....	41
6.10 Implementing the application using JADE .....	43
6.11 Summary .....	44
<b>Chapter 7 Evaluation.....</b>	<b>45</b>
7.1 Introduction.....	45
7.2 Evaluating the route match algorithm.....	45
7.3 MAS Simulation .....	46
7.4 Simulation Results .....	47
7.5 Summary .....	51
<b>Chapter 8 Conclusion .....</b>	<b>52</b>
8.1 Introduction.....	52

8.2	Discussion .....	52
8.3	Future work .....	54
8.4	Summary .....	55
<b>References .....</b>		<b>56</b>
<b>Appendix A: JADE Architecture Overview .....</b>		<b>60</b>
A.1	Introduction .....	60
A.2	JADE Architecture Overview .....	60
<b>Appendix B: List of simulated users and their routes .....</b>		<b>63</b>
B.1	Introduction .....	63
B.1	List of addresses for user simulation .....	63
B.1	List of simulated users and their routes .....	63
<b>Appendix C: FIPA ACL Message Structure .....</b>		<b>69</b>
C.1	Introduction .....	60
C.2	FIPA ACL message structure .....	69



University of Moratuwa, Sri Lanka.  
 Electronic Theses & Dissertations  
 www.ltu.mrt.ac.lk

## List of Figures

Figure 5.1: System architecture .....	26
Figure 5.2: Database Diagram .....	28
Figure 5.3: Store driver journey in system.....	29
Figure 5.4: Store passenger journey in system .....	30
Figure 5.5: Journey match process.....	31
Figure 6.1: Flow of agent behaviour.....	35
Figure 6.2: Haversine formula .....	39
Figure 6.3: Finding journey overlap .....	40
Figure 7.3: Rise is the number of optimal ride shares with number of simulated users ....	48



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

## List of Tables

Table 7.1: Optimal rideshare established for 50 users .....	47
Table 7.2: Number of rideshares against number of simulated users .....	48
Table 7.3: All possible dual member carpools for 100 users.....	50
Table 7.4: Carpool with 4 users .....	30
Table 7.5: Carpool with 3 users .....	51
Table 7.6: Carpool with 3 users .....	51



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

## Introduction

### 1.1 Prolegomenon

This thesis covers a research done to introduce a car pooling and ride sharing solution to Sri Lanka. It introduces a novel and generalized method towards matching passengers and drivers so that matching is based on journey intersections. Journey intersections are found by using a multi agent based technology which is supported by a maps ontology and API.

In Sri Lanka, especially in Colombo and its outskirts traffic congestion is increasingly evident most commonly during the morning when people are travelling to work, and again in the evenings when people are travelling back home. A significant portion of this traffic is caused by SOVs (Single Occupancy Vehicles). SOVs are not only a cause for unwarranted congestion but also contribute significantly to pollution on a per person basis. Quite often groups of co-workers from the same company/ institute live in adjoining neighborhoods. In such situations a car pooling/ride sharing solution will allow for these coworkers to travel to work together daily and thereby conserve valuable resources such as time and fuel. A worker who is riding to work via public transport, private transport (such as office van or shuttle service), will also have the opportunity to utilize the time spent travelling for a productive task, whereas a person driving does not have that luxury. Sharing rides or carpooling also has the promise to build strong social networks of support in society. Most importantly carpooling/ride sharing solutions will cut down on traffic congestion and pollution in the city.

### 1.3 Problem definition

Carpooling (also car-sharing, ride-sharing and lift-sharing), is the sharing of car journeys so that more than one person travels in a car. By having more people using one vehicle, carpooling reduces each person's travel costs such as fuel costs, tolls, and the stress of

driving. Carpooling is seen as a more environmentally friendly and sustainable way to travel as sharing journeys reduces carbon emissions, traffic congestion on the roads, and the need for parking spaces. Authorities often encourage carpooling, especially during high pollution periods and high fuel prices.

Carpooling is very popular in western countries such as the United States, Canada and many European countries. In such countries, carpooling is significantly correlated with transport operating costs, including gas prices and commute length. However, carpooling is significantly less likely among people who spend more time at work, older workers, and homeowners. Carpool commuting is more popular for people who work in places with more jobs nearby, and who live in places with higher residential densities[1].

Majority of car pools are “fam-pools” consisting of family members, however a car pool can also consist of office colleagues, neighbors or friends. Many new websites that offer carpooling solutions exploit social networks to find agreeable matches for carpools. Another variation of a carpool is a van/bus pool, where a high passenger capacity vehicle such as a van or bus is used to transport passengers along a certain predefined route. Van pools are quite popular in urban areas in Sri Lanka.

Carpooling is not always arranged for the whole length of a journey. Especially on long journeys, it is common for passengers to only join for parts of the journey, and give a contribution based on the distance that they travel. This gives carpooling extra flexibility, and enables more people to share journeys and save money.

Carpooling/ridesharing problem however has many challenges which need to be looked into when implementing a solution[2]. The main challenges are flexibility, reliability and security. Flexibility is an issue because riders in a carpool should agree to a fixed timeframe and route and cannot differ from that afterwards. Reliability becomes an issue mainly when setting up a carpool since if there isn't a system with 'critical mass' of participants it will be near impossible to find a plausible match for a carpool. Finally security becomes an issue when setting up carpools with total strangers. A popular



solution to security issues is the setting up of carpools via connections on social media networks.

#### **1.4 Aim and Objectives**

This research will attempt to provide an optimal and efficient multi agent based solution to finding journey matches between passenger and driver, based on finding journey intersections based on a map ontology. For this purpose real geospatial information will be extracted from a geographical information system. The journey matching will be conducted along the three aspects of spatial, temporal and social cultural requirements of the user. The system will also be built considering the ideal user interface requirements and will also consider situational requirements of the user. Finally the research will provide a simulation to showcase the functionality of the system.

#### **1.5 Outline**

This thesis gives a detailed description of the research carried out towards solving the problem of carpooling and ridesharing and achieving the above mentioned aims and objectives. Chapter 2 covers an in depth literature review into the problem domain and gives a study of previous research into the use of multi agent based systems in the transportation domain. It also looks into other intelligent solutions used to solve the ridesharing problem and conducts a brief comparison of these methods. Next, Chapter 3 focuses on the technology used to implement the solution and describes the multi agent system technology in detail, with special emphasis placed on the tools used for developing the multi agent system. Chapter 4 presents a detailed look into the approach taken in the research towards implementing a multi agent based solution for the problem. The next chapter, Chapter 5, presents the system design and a description of the system components. Chapter 6 gives a detailed look into the system implementation and explains the algorithms used and the system implementation using data flow diagrams and class diagrams. The following chapter, Chapter 7 describes the methods used for system evaluation. Finally Chapter 8 presents the conclusion of the thesis, the research

achievements, the issues encountered during the research process and the possibilities for extending this research in the future.

## **1.6 Summary**

This chapter gave a basic introduction to the research problem of developing a multi agent system for an optimal ride sharing solution. It covered the problem definition, the motivation and the research scope. Finally it also presented the thesis outline and a brief description of the chapters contained in the rest of the thesis. The next chapter will present the literature survey that covers the study of the problem and research carried out in this area.



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

## Chapter 2

### Analysis of carpooling/ridesharing solutions

#### 2.1 Introduction

This section covers previous work carried out in the area of multi agent systems for ride sharing and carpooling. It gives an overview of research carried out in this arena. It outlines the challenges of implementing carpooling/ride sharing systems and covers previous research on solutions to the problem in some detail. This chapter also conducts a comparison of the available intelligent solutions and their technologies and finally justifies the selection of multi agent systems for this research.

#### 2.2 Research into the challenges of carpooling and ridesharing

Carpooling is recognized as an alternative to reduce congestion and pollution on roads. However it is often difficult to implement and sustain at city or organizational level. In the paper ‘Making Car Pooling Work – Myths and Where To Start’ [1], the researcher clarifies some popular myths in carpooling and suggests areas where organizations can work on to make carpooling more effective. The first myth the paper looks into is that finding drivers and riders will increase car pooling. The research points out that such efforts generally result in finding more riders than drivers and that even when matches are found there is no guarantee that the car pool will actually work. The researcher believes this is due to a missing alignment between driver and rider. The second myth looked into is that giving money to drivers will promote car pooling. The research points out that giving money won’t have much of an effect because drivers are often looking for a car pool where they can switch between driving and riding (ie. someone else will drive another day). The next myth looked into is that car pooling can be a low cost alternative to a shuttle. The researcher identifies that organizations are likely to prefer car pooling as a low cost alternative to setting up a shuttle service. However the researcher again points out that shuttle riders expect a timely service, however car poolers will be dependent on the schedule of the driver. Having debunked popular myths about car pooling the research continues to identify steps that can be taken, especially at organizational levels

to promote car pooling. Some of the steps given are, to identify compatible groups with similar commuting needs(with special attention paid to matching schedules), giving incentives to carpoolers (some cities have separate lanes for car poolers) and register and recognize car pools.

A variation of car pooling is dynamic car pooling, which takes advantage of the recent and increasing adoption of Internet-connected geo-aware mobile devices for enabling impromptu trip opportunities. Passengers request trips directly on the street and can find a suitable ride in just few minutes. There has been some research into identifying the most important issues against the adoption of dynamic carpooling systems and the proposed solutions for such issues[2]. The main barriers to adaptation of a dynamic car pooling system are identified as, poor system interface design, problems with driver-passenger matching algorithms, aspects related to how people meet, authenticate and coordinate, safety and trustworthiness, reaching critical mass (the amount of persons using the system that would attract more people) and incentives[2].



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
www.lib.mtu.ac.lk

In an article published about the barriers to adaptation of car pooling in India, the writer identifies consumer behavior as the reason for reluctance for adaptation of car pooling[3]. It is stated that especially in developing countries such as India, owning a vehicle symbolizes wealth and success and driving one's own vehicle is generally a method in which to establish status. The article continues to identify various aspects of modern culture which could be a barrier to adaptation of car pooling.

A reference to car pooling as a solution to traffic congestion in Sri Lanka can be found in the article '*High fuel prices: Is car-pooling an option?*'[4], published in the Sunday Times. The article looks at how car pooling can be implemented in Sri Lanka and identifies the requirement for a computerized system to facilitate car pooling in Sri Lanka.

### **2.3 Solving the carpooling problem using multi agent systems**

Computing ideal ridesharing plans is a challenging problem as the solution must consider the varied and dynamically changing preferences of self-interested agents, must provide

compelling and fair incentives, and must be easy to use [5]. The paper ‘Collaboration and Shared Plans in the Open World: Studies of Ridesharing’ introduces a system with three main components, a user-modeling component that accesses and represents the preferences of agents, an optimization component that generates collaborative rideshare plans, and a payment component that provides incentives to agents to collaborate. The user-modeling component employs a probabilistic time cost model. The model considers as input the time of day, day of week, and sets of attributes about agents’ commitments drawn from an online appointment book. Probabilistic models for the cost of time and for the commitment to attend events are learned from user annotated training data via a machine-learning procedure based on Bayesian structure search. Similar predictive models of the cost of time and meeting commitments have been used in other applications, including mobile opportunistic planning [6], meeting coordination [7] and the triaging and routing of communications [8]. The second component; optimization component takes in the set of individual desired commute plans as inputs and solves two difficult optimization problems to generate a collection of collaborative rideshare plans. The two optimizations are, generating rideshare plans for groups of agents and clustering agents into rideshare groups. In the third component a personal inconvenience cost that captures several agent-specific cost factors is considered. The personal inconvenience factors are composed to yield the cumulative value of a rideshare plan. Finally the paper also included a real world trip data set study [9] which showed increasing fuel costs as well as higher number of users increased the rideshare efficiency, whereas increased time cost decreased the efficiency.

In the paper ‘A Matching-Algorithm based on the Cloud and Positioning Systems to Improve Carpooling’ [10] a geosocial network is exploited to improve the users’ confidence in the rides arranged with other passengers. The route calculation algorithm is one of the key challenges of the proposed solution. The optimal solution is identified as an NP problem, requiring to compute all the possible arrangements of rides involving all the friends in the social network. As a consequence, some heuristics are looked into to make the problem addressable. The basic idea was to exploit the Cloud to search for different solutions, each of them uses a greedy approach on a different starting item. In particular, the approach explores up to 50 different solutions, differing in the people

involved in the computed ride. Each solution is elaborated on a different Worker in the Cloud. Once all of them have computed the total distance to cover, the best solution is picked. More in details, the resulting route calculation algorithm is based on the following five steps. In the first step computes the route for the user who set-up the trip, by invoking the Bing Maps Web Service on his/her starting and ending destinations. The service returns the path, intended as a sequence of maneuvers. The second step defines a circle around the starting point of the route, whose radius is defined according to the user preference about the maximum allowance for a detour. The third step looks for friends (and then friends of friends) within the social network connections, whose departure point fall within the above defined circle. If there is a candidate, the new route to match friend's destination is computed. If the detour is bigger than driver's preferences, the friend is discarded, and another solution is searched for. If more than one candidate is found, the different solutions are computed on different "workers" on the Cloud, beginning from the closer one. The search is limited to the 50 candidates whose starting points are close to the driver's one. Among all the found solutions, the one which minimizes the detour is selected. In the fourth step, if the computed detour is shorter than the maximum allowed by the driver, the next potential travel-mate is considered, moving the center on the friend's starting point and reducing the radius of the research consistently. In the fifth step iteration is done until (I) the maximum number of passenger is reached, or (II) the maximum detour distance has been reached, or (III) no feasible solution is found. In the latter case, the system asks the user if he/she is interested in involving unknown people in the search. Further heuristics are included in the algorithm, suited to prefer friends rather than friends of friends in the arrangement of the ride.

The publication 'Genghis - A Multiagent Carpooling System'[11] gives a detailed description of a multi agent based system for car pooling. This system like many others solves the journey matching problem by concentrating on journey start and end points only. It doesn't offer a solution for partial journey matching. The system described consisted of the following agents, UserAgent (represents a human user and their allowed interaction with Genghis), ProxyAgentN (this agent will service HTTP requests and act as the middleware between a Jade container and web application), JourneyRoundupAgent (flags journeys which are past their end time for human feedback),

JourneyNotifyAgent(keeps watch on the wanted and active journeys and flags UserAgents if any match comes about).

Agent based modeling (AgnBM) simulates interactions between individuals in order to assess the effect on the society as a whole. The paper ‘Analysis of the Co-routing Problem in Agent-based Carpooling Simulation’[12], uses AgnBM to investigate interaction between carpooling people. The agentBased model simulates between 1000 and 5000 individuals belonging to the synthetic population generated for Flanders (Belgium). This amount of agents is sufficient to investigate the carpooling phenomenon and is expected to be small enough to keep the problem computationally tractable. A social network joining the agents is built and evolves as described in [13],[14]. Small sets of agents (typically 2 to 5) negotiate route choice and travel time in order to carpool e.g. for commuting on a specific day of the week. Schedule execution is simulated and introduces stochastic deviations between the actual and planned schedule versions. Behaviourally relevant factors such as VOT (value of time) and time use flexibility are involved. The model is used to evaluate both the effect of, (a) travel-parking costs and carpool parks availability on the overall travel demand and (b) the complexity of the drivers cooperation process itself as an inhibiting factor (due to required schedule adaptation). Carpooling candidates explore their social networks in order to detect possible fellow travellers and negotiate a route (coRouting) which requires schedule adaptation (reScheduling). Key components are exploration, negotiation (requiring coRouting and reScheduling) and schedule execution. Those are coordinated by the agentBased model. Rescheduling involves shifting activities (and hence travel) in space-time using limited activity reordering and making use of VOT, disutility functions and lists of feasible locations for activity execution. CoRouting includes route choice and mode selection (walk, bike, car, public transportation) and affects route duration but not absolute time (trip start time). CoRouting and reScheduling thus are orthogonal concepts: they can be studied independently. By negotiating, each agent tries to minimize their total cost which is the sum of travel cost and schedule adaptation disutility cost. Each passenger pays a weighted part of the drivers original trip distance cost plus a weighted part of the excess generalized cost for the driver caused by trip distance and duration increase. Both coRouting and reScheduling involve frequent solution of moderately sized



optimisation problems. The paper continues to detail a graph based solution to the coRoutingsubproblem.

The paper 'An Agent Solution to Flexible Planning and Scheduling of Passenger Trips'[15] presents the MADARP agent architecture, devoted to the planning and scheduling of trip requests under a dynamic scenario within the context of passenger transportation systems. The architecture provides a set of base agents that perform the basic interface, planning and support services for managing different types of transportation requests by using a heterogeneous fleet of transport vehicles. The architecture was used to implement three planning models by extending base agents' behaviors. The results obtained for a set of 20 scenarios was then analyzed. The agent architecture is built-up over the Jade agent platform [16], which provides a distributed environment organized in containers where agents can work, communicate and migrate within them. The MADARP agent architecture [17] consists of four layers that group the agents and structures according to the functionality provided. The Interface layer connects the system with the real world; the Planning layer performs the trips processing; and the Service layer provides different complementary functionalities. At the bottom the Service Ontology provides a means to integrate and make interacting the different agents and actors from the upper layers in a transparent and coherent way. The system described in the paper consisted of the following agents, the interface layer consisted of vehicle and client agents. The planning layer consisted of the scheduling agent (coupled with vehicle agent), the trip-request agent (coupled with the client agent) and the planner agent. The service layer consisted of broker, map, account, traffic and payment agents.

#### **2.4 Other solutions to the carpooling problem**

Car pooling can be categorized into two different forms, Dynamic Car Pooling Problem (DCPP) and Long Term Car Pooling (LTCPP). For DCPP, on each day a number of users are available for picking up or bringing back their colleagues in that particular day. For LTCPP, each user has to act as both a server and a client; the objective is to define user pools where each user will pick up the remaining pool members in turn, on different days.



The paper ‘A Decision-Support System for the Car Pooling Problem’[18] specifically addresses the LTCPP. It follows a three step process of data collection, clustering users to groups and vehicle routing. The clustering mechanism works by calculating the similarity matrix for different routes via a heuristics based pearson correlation coefficient calculation. Once the clustering is done a genetic algorithm based solution is used to solve the travelling salesman problem (ie. finding the optimal route) for each cluster.

The paper ‘Safe Ride’[19] considers a graph based solution to matching drivers with passengers in a car pooling problem. Conceptually, you can think of the data structure as a three dimensional grid with each horizontal plane representing the physical region and the vertical dimension representing time. Each planned driver’s trip is a line through the space, ascending through time. The space can be made into a discreet graph by choosing five minute intervals and forcing each line to go through street intersections. Then finding a ride for someone amounts to a solving a single-source-destination shortest path problem. Because the graph is very sparse and other considerations, the actual data structure is a directed graph of every intersection where a pick-up or drop-off could occur. Permanent edges to adjacent intersections are labeled "driving", "walking", "bicycling", etc. along with transit times. There is no third dimension; instead each node has a time-ordered list of expected arrivals of drivers. The paper proposes that when a driver enters, plot their route using a shortest path algorithm, allow them to modify, and insert them in the arrival schedule at every node along the route, and when a rider enters, find the match by performing a shortest path search in which the edges between vertices are actual drivers traveling between those vertices at an appropriate time. The paper also emphasizes the importance of pilots for success of a proposed system and suggests that universities be used as pilot grounds.

The paper ‘Exploiting Graph-theoretic Tools for Matching in Carpooling Applications’[20] gives a detailed look into a graph theory and multi agent based solution for the carpooling problem. Here planned periodic trips correspond to nodes in a graph; the edges are labeled with the probability for success while negotiating to merge two planned trips by carpooling. The probability values are calculated by a learning mechanism using on one hand the registered person and trip characteristics and on the

other hand the negotiation feedback. The probability values vary over time due to repetitive execution of the learning mechanism. As a consequence, the matcher needs to cope with a dynamically changing graph both with respect to topology and edge weights. In order to evaluate the matcher performance before deployment in the real world, the research includes a large scale agent based model. The paper describes in detail both the exercising model and the matcher.

## 2.5 Summary

This chapter presented a study of previous work conducted towards implementing an intelligent solution for the carpooling/ridesharing problem. It first looked into the various challenges raised by the problem and into research carried out towards solving those challenges. Next it gave a detailed study of the use of multi agent systems for implementing a carpooling and ridesharing solution. This section covered various previous such implementations and identified their limitations. In particular the lack of a solution for the joint route matching problem was identified. Finally the chapter looked into other solutions to the carpooling/ridesharing problem. These included genetic algorithm based solutions and decision support systems. In this section the superiority of a multi agent based system over these technologies was discussed. The following chapter will discuss the technology used in this research and the reasons for its selection.

## Chapter 3

# Multi Agent Systems and their applications

### 3.1 Introduction

This section details the technology used in this research. Given that previous research in the area of carpooling and ride sharing mechanisms was heavily based on multi agent systems, this research also adapts a multi agent solution. This chapter gives a high level overview on multi agent systems, their architecture and practical use cases.

### 3.2 Agent Definition

The definition of an agent is subject to argument, however in general an agent can be perceived as a small computer program that activates when necessary, completes a specific task and then terminates.

Russel&Norvig [21] group agents into five classes based on their degree of perceived intelligence and capability.



University of Moratuwa, Sri Lanka.

Electronic Theses & Dissertations

[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

#### 1. Simple reflex agents

Simple reflex agents act only on the basis of the current percept, ignoring the rest of the percept history. The agent function is based on the *condition-action rule*: if condition then action.

This agent function only succeeds when the environment is fully observable. Some reflex agents can also contain information on their current state which allows them to disregard conditions whose actuators are already triggered.

## 2. Model-based reflex agents

A model-based agent can handle a partially observable environment. Its current state is stored inside the agent maintaining some kind of structure which describes the part of the world which cannot be seen. This knowledge about "how the world works" is called a model of the world, hence the name "model-based agent".

A model-based reflex agent should maintain some sort of internal mode that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state. It then chooses an action in the same way as the reflex agent.

## 3. Goal-based agents

Goal-based agents further expand on the capabilities of the model-based agents, by using "goal" information. Goal information describes situations that are desirable. This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state. Search and planning are the subfields of artificial intelligence devoted to finding action sequences that achieve the agent's goals.

In some instances the goal-based agent appears to be less efficient; it is more flexible because the knowledge that supports its decisions is represented explicitly and can be modified.

## 4. Utility-based agents

Goal-based agents only distinguish between goal states and non-goal states. It is possible to define a measure of how desirable a particular state is. This measure can be obtained through the use of a *utility function* which maps a state to a measure of the utility of the state. A more general performance measure should allow a comparison of different world states according to exactly how happy they would make the agent. The term utility, can be used to describe how "happy" the agent is.

A rational utility-based agent chooses the action that maximizes the expected utility of the action outcomes- that is, the agent expects to derive, on average, given the probabilities and utilities of each outcome. A utility-based agent has to model and keep

track of its environment, tasks that have involved a great deal of research on perception, representation, reasoning, and learning.

## 5. Learning agents

Learning has an advantage that it allows the agents to initially operate in unknown environments and to become more competent than its initial knowledge alone might allow. The most important distinction is between the "learning element", which is responsible for making improvements, and the "performance element", which is responsible for selecting external actions.

The learning element uses feedback from the "critic" on how the agent is doing and determines how the performance element should be modified to do better in the future. The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.

The last component of the learning agent is the "problem generator". It is responsible for suggesting actions that will lead to new and informative experiences.



University of Moratuwa, Sri Lanka  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

### 3.3 Multi agent system implementation

Given the definition of an agent, a multi agent system can be defined as a computerized system consisting of multiple agents within a specific environment, working towards a common purpose. Three key features in a multi agent system are communication, coordination and negotiation between agents.

While ad hoc multi-agent systems are often created from scratch by researchers and developers, some frameworks have arisen that implement common standards (such as the FIPA[22] agent system platforms and communication languages). These frameworks save developers time and also aid in the standardization of MAS development.

### 3.4 Multi agent system architectures

Agent architectures range from purely reactive (or behavioural) architectures that operate in a simple stimulus–response fashion, such as those based on the *subsumption architecture* of Brooks[23] , to more deliberative architectures that reason about their actions, such as those based on the belief desire intention (BDI) model[24]. In between the two lie hybrid combinations of both, or layered architectures, which attempt to involve both reaction and deliberation in an effort to adopt the best of each approach. Thus agent architectures can be divided into four main groups: logic based, reactive, BDI and layered architectures.

1. *Logic-based* (symbolic) architectures draw their foundation from traditional knowledge-based systems in which an environment is symbolically represented and manipulated using reasoning mechanisms. The advantage of this approach is that human knowledge is symbolic so encoding is easier, and they can be constructed to be computationally complete, which makes it easier for humans to understand the logic. The disadvantages are that it is difficult to translate the real world into an accurate, adequate symbolic description, and that symbolic representation and manipulation can take considerable time to execute with results being often available too late to be useful.
2. *Reactive* architectures implement decision-making as a direct mapping of situation to action and are based on a stimulus–response mechanism triggered by sensor data. Unlike logic-based architectures, they do not have any central symbolic model and therefore do not utilize any complex symbolic reasoning. Probably the best-known reactive architecture is Brooks's subsumption architecture[23]. The key ideas on which Brooks realized this architecture are that an intelligent behaviour can be generated without explicit representations and that intelligence is an emergent property of certain complex systems. The subsumption architecture defines layers of finite state machines that are connected to sensors that transmit real-time information.

3. BDI (Belief, desire, intention) architectures are probably the most popular agent architectures[24]. They have their roots in philosophy and offer a logical theory which defines the mental attitudes of belief, desire and intention using a modal logic. Many different agent-based systems have been realized that implement BDI with a wide range of applications demonstrating the viability of the model. One of the most well-known BDI architectures is the Procedural Reasoning System (PRS)[25]. This architecture is based on four key data structures: beliefs, desires, intentions and plans, and an interpreter.
4. *Layered* (hybrid) architectures allow both reactive and deliberative agent behaviour. To enable this flexibility, subsystems arranged as the layers of a hierarchy are utilized to accommodate both types of agent behaviour. There are two types of control flows within a layered architecture: horizontal[26] and vertical layering[27].

**3.5**  **Communication, coordination and negotiation**  
University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

One of the key components of multi-agent systems is communication. In fact, agents need to be able to communicate with users, with system resources, and with each other if they need to cooperate, collaborate and negotiate. In particular, agents interact with each other by using some special communication languages, called agent communication languages, that rely on speech act theory[28]. The first agent communication language with a broad uptake was KQML[29]. KQML was developed in the early 1990s as part of the US government's ARPA Knowledge Sharing Effort. It is a language and protocol for exchanging information and knowledge that defines a number of performative verbs and allows message content to be represented in a first-order logic-like language called KIF[30]. Currently the most used and studied agent communication language is the FIPA, which incorporates many aspects of KQML[31]. The primary features of FIPA ACL are the possibility of using different content languages and the management of conversations through predefined interaction protocols.



Coordination is a process in which agents engage to help ensure that a community of individual agents acts in a coherent manner[32]. There are several reasons why multiple agents need to be coordinated including: (1) agents' goals may cause conflicts among agents' actions, (2) agents' goals may be interdependent, (3) agents may have different capabilities and different knowledge, and (4) agents' goals may be more rapidly achieved if different agents work on each of them. Coordination among agents can be handled with a variety of approaches including organizational structuring, contracting, multi-agent planning and negotiation.

Organizational structuring provides a framework for activity and interaction through the definition of roles, communication paths and authority relationships[33]. The easiest way of ensuring coherent behaviour and resolving conflicts seems to consist of providing the group with an agent which has a wider perspective of the system, thereby exploiting an organizational or hierarchical structure. This is the simplest coordination technique and yields a classic master/slave or client/server architecture for task and resource allocation among slave agents by a master agent. The master controller can gather information from the agents in the group, create plans and assign tasks to individual agents in order to ensure global coherence. However, such an approach is impractical in realistic applications because it is very difficult to create such a central controller, and in any case, centralized control, as in the master/slave technique, is contrary to the decentralized nature of multi-agent systems.

An important coordination technique for task and resource allocation among agents and determining organizational structure is the *contract net* protocol[34]. This approach is based on a decentralized market structure where agents can take on two roles, a manager and contractor. The basic premise of this form of coordination is that if an agent cannot solve an assigned problem using local resources/expertise, it will decompose the problem into sub-problems and try to find other willing agents with the necessary resources/expertise to solve these sub-problems. The problem of assigning the sub-problems is solved by a contracting mechanism consisting of: (1) contract announcement by the manager agent, (2) submission of bids by contracting agents in response to the



announcement, and (3) the evaluation of the submitted bids by the contractor, which leads to awarding a sub-problem contract to the contractor(s) with the most appropriate bids.

Another approach is to view the problem of coordinating agents as a planning problem. In order to avoid inconsistent or conflicting actions and interactions, agents can build a multi-agent plan that details all the future actions and interactions required to achieve their goals. Multi-agent planning can be either centralized or distributed. In *centralized multi-agent planning*, there is usually a coordinating agent that, on receipt of all partial or local plans from individual agents, analyses them to identify potential inconsistencies and conflicting interactions (e.g. conflicts between agents over limited resources). The coordinating agent then attempts to modify these partial plans and combines them into a multi-agent plan where conflicting interactions are eliminated [35]. In *distributed multi-agent planning*, the idea is to provide each agent with a model of other agents' plans. Agents communicate in order to build and update their individual plans and their models of other agents until all conflicts are removed [36]. Partial global planning integrates the strengths of the organizational, planning, and contracting approaches by uniting them into a single approach [37]. The goal of this approach is to gain the multi-agent planning benefits of detailed, situation-specific coordination while avoiding excessive computation and communication costs. This is possible because the jointly known organizational structures effectively prune the space of possible plans to keep the problem tractable.

Negotiation is probably the most relied upon technique for coordinating agents. In particular, negotiation is the communication process of a group of agents in order to reach a mutually accepted agreement on some matter [38]. Negotiation can be competitive or cooperative depending on the behaviour of the agents involved. Competitive negotiation is used in situations where agents have independent goals that interact with each other; they are not a priori cooperative, share information or willing to back down for the greater good. Cooperative negotiation is used in situations where agents have a common goal to achieve or a single task to execute. In this case, the multi-agent system has been centrally designed to pursue a single global goal.

### 3.6 JADE (Java Agent Development Framework)

JADE is a FIPA compliant agent development framework written in Java. It is available as open source software and is the popular choice in multi agent system development, therefore it is chosen as the implementation framework for this research project. JADE is a middleware that facilitates the development of multi-agent systems. It includes aruntime environment where JADE agents can “live” and that must be active on a given host before one or more agents can be executed on that host, a library of classes that programmers have to/can use (directly or by specializing them) to develop their agents. A suite of graphical tools that allows administrating and monitoring the activity of running agents.

Each running instance of the JADE runtime environment is called a *Container* as it can contain several agents. The set of active containers is called a *Platform*. A single special *Main container* must always be active in a platform and all other containers register with it as soon as they start.

Besides the ability of accepting registrations from other containers, a main container differs from normal containers as it holds two special agents (automatically started when the main container is launched). The AMS (Agent Management System) that provides the naming service (i.e. ensures that each agent in the platform has a unique name) and represents the authority in the platform (for instance it is possible to create/kill agents on remote containers by requesting that to the AMS). The DF (Directory Facilitator) that provides a Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals[23].

### 3.7 MAS in Traffic and Transportation

‘Applications of multi agent systems in traffic and transportation’[40] describes applications of MAS in traffic and transportation, beginning with an overview on AOT (Agent Oriented Techniques) and the BDI paradigm. The areas identified where MAS will have an impact are in the analysis and description of traffic systems, increasing the autonomy of traffic components and an integration framework (for example an

Emergency Rescue Management centre that links up accident, pollution and decision support modules). The DASEDIS architecture is described to model traffic, based on a BDI model. We see models of agent behaviour when cars have to overtake each other and free driving. Carsharing is covered, where agents represent stations and customers. Other than these details, aspects of methodology are not covered.

### **3.8 Summary**

This chapter gave a detailed look into the selection of the technology used for this research, which is multi agent based technology. Here the various aspects of multi agent based systems were identified. In particular the different multi agent based system architectures available were discussed and a suitable system architecture was chosen. Finally an implementation framework for the system was chosen based on the selected multi agent system architecture. The next chapter presents an over view of the approach taken towards solving the carpooling/ridesharing problem using the selected multi agent based technology.



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

## Chapter 4

### Solving complexity using a multi agent system

#### 4.1 Introduction

This gives a detailed description on the approach taken to solve the research issue of implementing a ridesharing/carpooling system in Sri Lanka. As identified in the ‘Technology’ chapter, the solution is a multi-agent system based approach.

#### 4.2 Hypothesis

The proposed solution is to implement a multi agent system that can match users for rideshares and carpools on the system. An agent based solution is considered to be effective due to the complexity of the problem, since it is required to match based on time (temporal) and route (spatial) requirements. In addition personal preferences must also be considered. A multi agent based system would be able to provide the three key elements required for the success of a ride sharing/carpooling system. Those are communication, negotiation and coordination among the drivers and the passengers. In particular this solution will look towards implementing an algorithm/logic that considers the travel routes/roads much the same way in which a human will try to solve the problem as opposed to considering only location points. The idea is to find overlapping routes of users and accordingly establish optimal rideshares or carpools. Optimality is measured based on the total travel distance of the shared route. A rideshare with a higher route coverage is considered more optimal than a ride share with lesser route coverage.

#### 4.3 System requirements and features

A carpooling/ridesharing solution for Sri Lanka should meet the following requirements,

- Allow users to register on the system and provide their details. A user will specify whether they are looking for a rideshare or carpool. Carpools can only be established between users who are vehicle owners.
- Allow users to setup trips on the system specifying start and destination and expected departure and arrival times. They should also be able to schedule recurring trips on the system.
- Allow users to specify their required ride on the system and find a match from the pool of registered user routes on the system.
- Display overlays of paths on a map when journey matches are found.

#### 4.4 Solution overview

##### Input

The system input is twofold. Firstly there is the user input data. In this case the input is the user details and the planned journey details. The user details includes basic user information, their social connection (to other users) and most importantly whether they are registering as a driver or as a passenger on the system. A passenger can be identified as a user who does not own a vehicle and is only looking for a rideshare. A driver on the other hand can be identified as a user who owns a vehicle and therefore can participate in both rideshares and carpools. In addition user input also included the specific planned journeys of the users. The journey information is both spatial and temporal. Spatial information is simply given as the start, end locations (addresses) of the journeys. The temporal information is simply the planned departure and arrival times for the journeys.

The second input type is the map data or geolocation data that is retrieved from the Google Maps API. This information is based on the planned journey inputs given by the users. The Google Maps API is accessed and the relevant geo location data for these journeys are retrieved via the API. This geolocation data consists of latitude and longitude points that lie along the routes given as expected journeys. The Google Maps

API returns this information encoded as polylines in order to compress the returned amount of data.

## Output

The system output would be the corresponding driver matches for a passenger user. This output will be depicted as the number of overlapping routes between the specific passenger and other registered driver users. The results can be ranked based on social connections with drivers having closer social connections to passengers having a higher ranking. The available matches are displayed on a map so that the passenger has a quick overview of all driver options available for a journey.

## Process

The process of converting the system inputs into the desired output is based on the multi agent system architecture of the system. First the system interface layer will receive the user inputs and propagate them to the agent and ontology layers. The agent layer will consist of the corresponding agents who will model driver and passenger agents. It will also consist of the route match and maps agents who will process journey match requests using the route match algorithm and geo location data for the specific routes. When required the agent layer will interact with the ontology layer to retrieve the information necessary for the route match process. Once the process completes the results can then be displayed to the system users as the system output.

## 4.5 Summary

This chapter gave a description on the approach taken in this research to solve the carpooling/ride sharing problem in the country. It gave a detailed description of the research hypothesis and the system input, output and process. It paves the way for the next chapter which describes the system design based on the solution approach.

## Chapter 5

### Design

#### 5.1 Introduction

This section details the solution design. It gives the design of the overall system and its architecture. In particular it identifies the system modules and explains the system use cases. This chapter precedes the chapter giving the system implementation and therefore sets the foundation towards the system implementation.

#### 5.2 System layered architecture

The system has three layers. The interface layer is a web based interface where users can register and enter their route details. The interface layer closely interacts with the agent layer. The agent layer consists of passenger, driver and route match agents. Here a passenger agent will send call for proposals to the driver agents. The driver agents will then contact a route match agent to check if the journey schedule, routes and personal preferences match. If a match is possible the route match agent will inform this to the driver agent. The driver agent will then reply back to the passenger agent proposing or rejecting a ride-share. The final layer is the ontology layer consisting of the map module, data store and social network module. This layer is contacted by the agents in the agent layer to get the information they need to function. Figure 5.1 given below presents the layered architecture of the system.

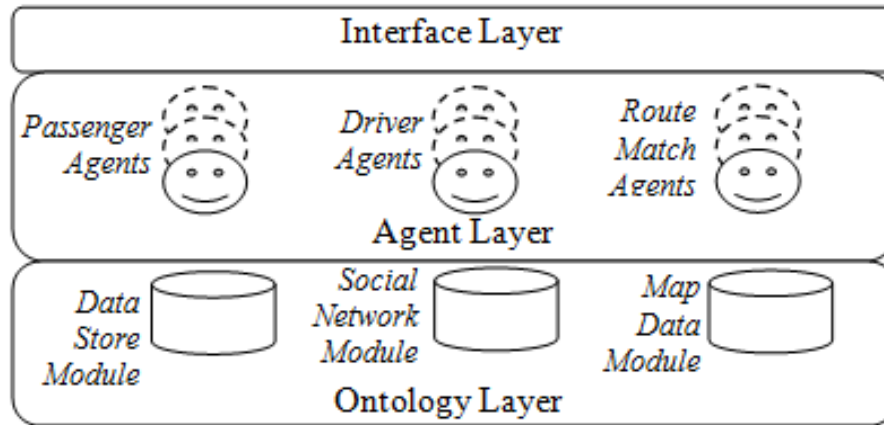


Figure 5.1: System architecture

### 5.3 Interface Layer

The interface layer consists of the user registration page and the journey scheduling page which includes a Google Map view which will display the scheduled and matched journeys to the user.

### 5.4 Agent Layer



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

- **Passenger Agent:** The passenger agent represents a system user who searched for a journey match as a passenger. The task of the passenger agent is to send requests for proposals to all driver agents.
- **Driver Agent:** The driver agent represents a system user who is offering a ride or looking for a car pool on the system. The task of the driver agent is to accept proposals from a passenger agent and to respond to that proposal based on whether or not the proposal can be met. The driver agent does this by sending out requests to the route match agent to determine if the passenger's route is a match to the driver's route.
- **Route Match Agent:** This agent is the brain behind the system. It uses the route matching algorithm to determine if the routes of a passenger and driver agent match. It responds back to the driver agent acknowledging or denying the match.



- Map Agent: This agent fetches route information needed by the route match agent by calling the Google Maps API. The retrieved route information is then passed back to the calling route match agent.

## 5.5 Ontology Layer

- Map Module: The map module connects with the Google Maps Directions API webservice to retrieve route information required for determining route matches. Here each passenger's or driver's start and destination information is used to fetch the geo coded location points (latitude, longitude pairs) of their routes. This information can then be fed to the Route Match Agent for finding journey matches.
- Social Network Module: The social network module connects with a social network to determine the social connections between the driver and passenger agents and thereby contributes to the security and reliability of the carpooling system. Here Facebook is considered to be the ideal social network since it is widely used across all generations in Sri Lanka. The social network module can either be connected with Facebook or can simulate social connections for the purpose of a carpooling simulation. In case of such a simulation a fair assumption of each passenger or driver agent having between 1-5 connections with other agents is considered.
- Data Store Module: This module acts as the data storage for the system. Given below in Figure 5.2 is the system database diagram.

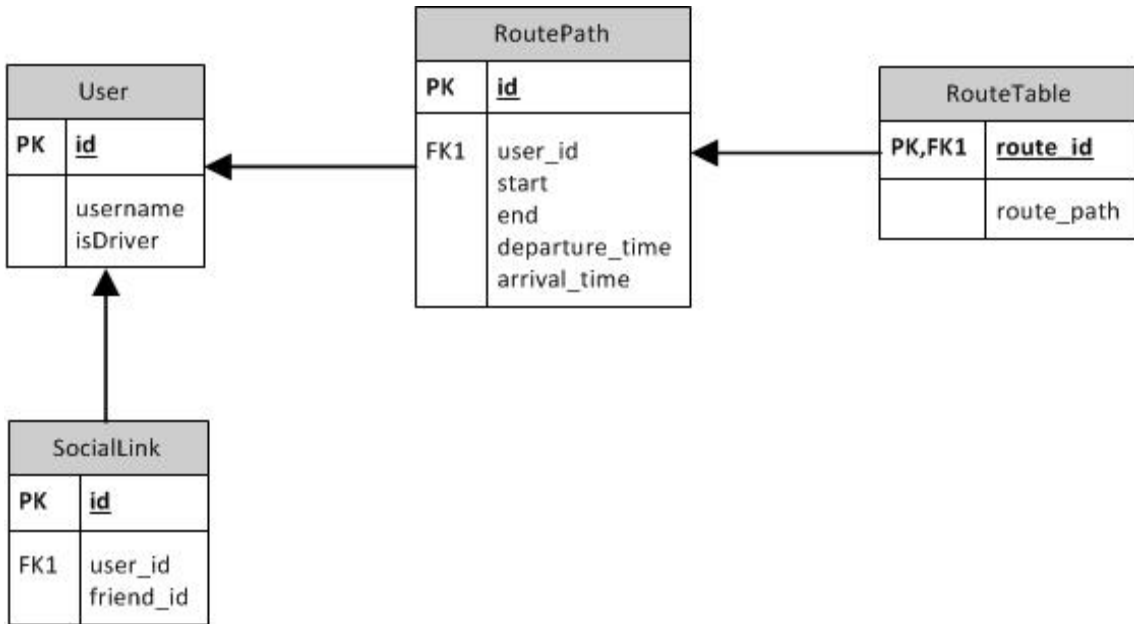


Figure 5.2: Database Diagram

The system database consists of four entities. The user entity table is used for storing all user information. This includes both driver and passenger users. A field ‘isDriver’ is used for the purpose of differentiating driver users from passenger users. The next entity is the routepath entity. This table tracks the journey routes for each user. It store the spatial and temporal data corresponding to each journey. The route table on the other hand stores specific route information such as geo location coordinates for a specific route. The final entity, the social link entity gives the social connection between system users.

## 5.6 System activity diagrams

Given below are the system activity diagrams. In these activity diagrams the system users are modelled as agents and the interactions between these agents are depicted. In a traditional activity diagram the users depicted are the actual system users only. However here it is required to show the interactions between different agents in the system.

## Activity 1

The below activity diagram Figure 5.3 depicts a user adding a journey into the system as a driver.

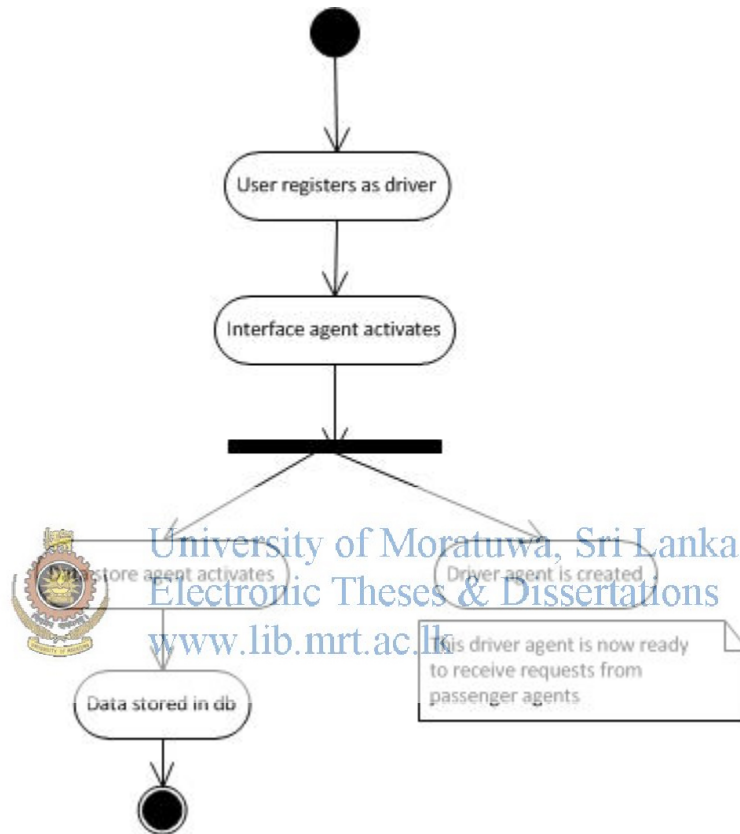


Figure 5.3: Store driver journey in system

This activity diagram shows how the interface agent is activated upon user request. The interface agent then forks off two processes. The first process activates a datastore agent who stores the user data in the system and the second process generates a driver agent corresponding to the user who registered as a driver on the system.

## Activity 2

A passenger requests a journey on the system. Here two tasks are executed in parallel. The journey is stored on the system and the system is searched for a driver who can fulfil the request.

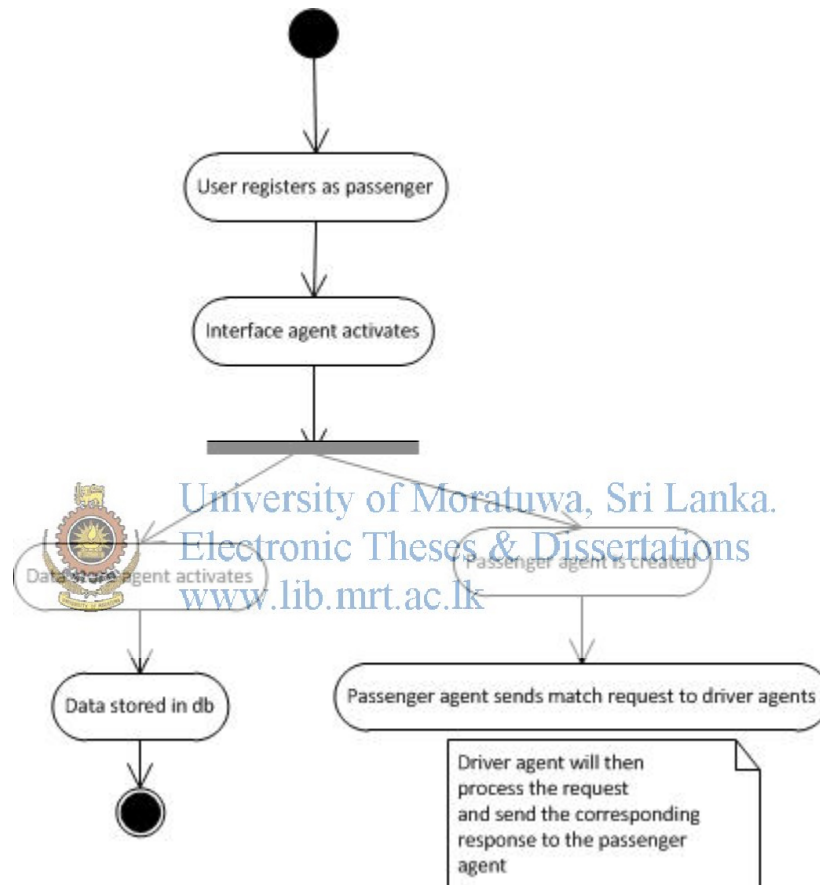


Figure 5.4: Store passenger journey in system

The flow of this activity is similar to that of the driver registration process. The only point of divergence is when the passenger agent is created a route match request is generated and sent to all driver agents currently active in the system.

### Activity 3

This activity depicts the search for a journey match.

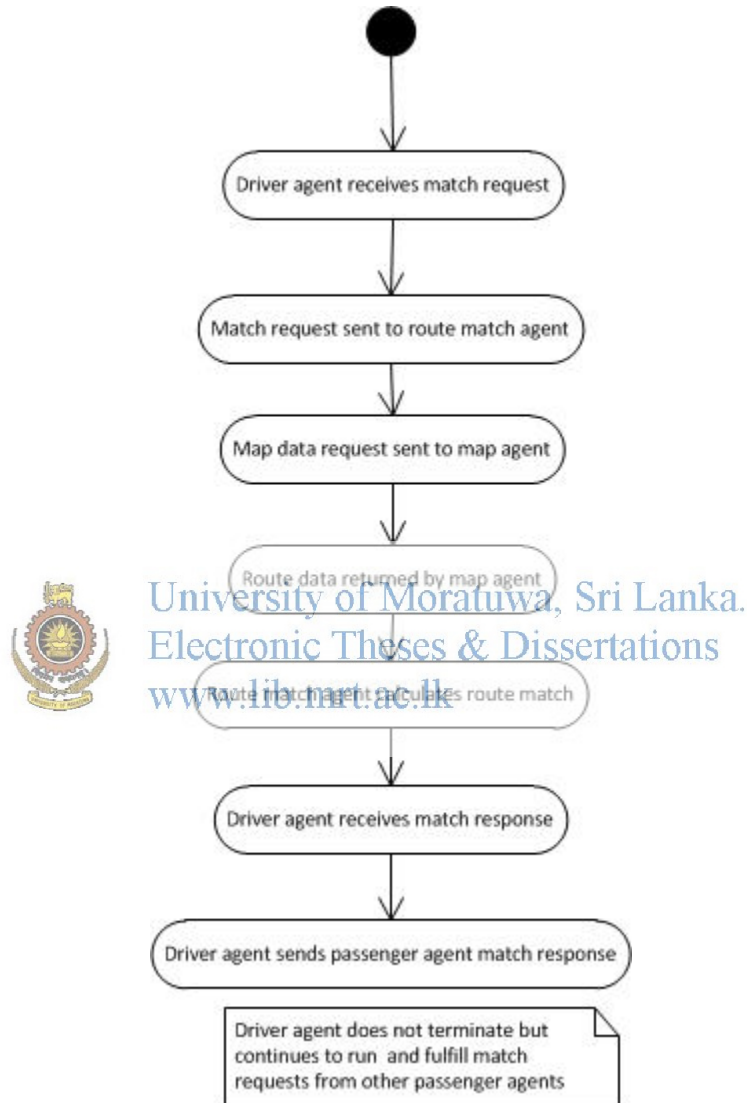


Figure 5.5: Journey match process

The journey match process runs between the passenger, driver, route match and maps agents. Although the process is sequential as per the activity diagram, since there are multiple passenger and driver agents in the system at any given time the same process runs parallel amongst the existing agents in the system. The process starts when a driver

agent receives a match request from a passenger agent. The driver agent then sends out a request to the route match agent to determine if the passenger is a compatible match. On receiving this request the route match agent sends out a corresponding request to the maps agent. The map agent uses the information received to retrieve the relevant route information from the maps ontology which is achieved via the Google Maps API. This information is then passed on to the route match agent. The route match agent then uses the information received to conclude whether the respective routes of the passenger and driver agent match. The route match agent then sends out a response to the driver agent confirming or refuting the match. The driver agent then send the appropriate response out to the passenger agent. As mentioned before this process happens in parallel between the different agents in the system based on the incoming passenger requests.

## 5.7 Summary

This section covered the system design. It presented the layered architecture of the system and a description of each layer and the modules and components of those layers. It also presented the system activity diagrams and an explanation of the system activity flow. The proceeding chapter describes the system implementation in detail.

## Chapter 6

### Implementation

#### 6.1 Introduction

This section details the project implementation. It explains in detail the system algorithm and the system implementation using JADE. This chapter closely relates to the previous chapter on system design.

#### 6.2 Interface module implementation

The interface module is the system interface. It is developed as a Java web application. This allows it to seamlessly communicate with the backend JADE implementation of the agent layer. The system interface is built using HTML/CSS/JSP/Javascript scripting and webpage technologies. In addition it also utilizes the Google Maps Javascript API to render map views in the interface. This allows users to view matched journey routes on a map of the Western Province of Sri Lanka.

#### 6.3 Agent layer implementation

The system agent layer is built by extending the JADE agent implementation. As mentioned in the design chapter the main agent types in the system are driver agents, passenger agents and route match agents. Note that there can be multiple of these types of agents interacting at any given time in the system.

#### 6.4 JADE agent behaviors

When implementing the agents it was required to follow predefined JADE agent behaviours for the purpose of implementing the functionality of specific agents. Given next is a description of these agent behaviours and the details of the corresponding agents that implemented these behaviours and the functional requirement for the implementation.

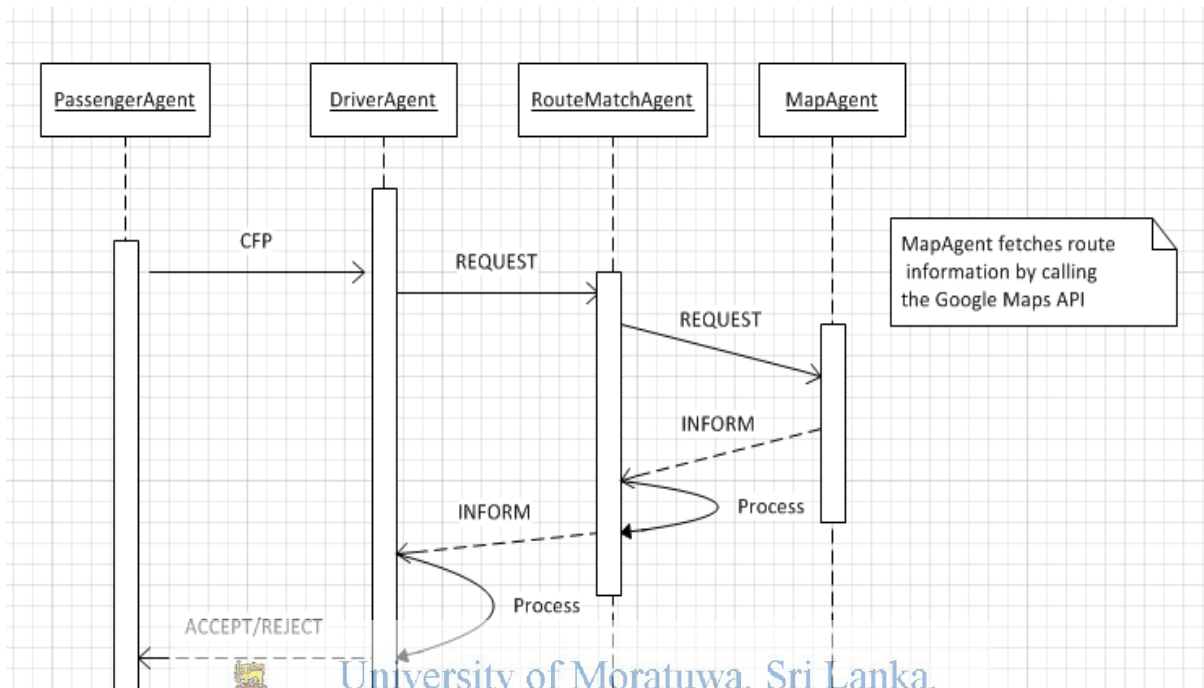
An agent can execute several behaviours concurrently. However it is important to notice that scheduling of behaviours in an agent is not pre-emptive (as for Java threads) but cooperative. This means that when a behaviour is scheduled for execution its action() method is called and runs until it returns. Therefore it is the programmer who defines when an agent switches from the execution of a current behaviour to the execution of the next one. Though requiring a small additional effort to programmers, this approach has several advantages.

- Allows having a single Java thread per agent (that is quite important especially in environments with limited resources).
- Provides better performances since behaviour switch is extremely faster than Java thread switch.
- Eliminates all synchronization issues between concurrent behaviours accessing the same resources (this speed-up performances too) since all behaviours are executed by the same Java thread.
- When a behaviour switch occurs the status of an agent does not include any stack information and is therefore possible to take a “snapshot” of it. This makes it possible to save the status of an agent on a persistent storage for later resumption.

The agent behaviours modelled in the system are Cyclic Behaviour, Sequential Behaviour and SimpleBehaviour. Cyclic Behaviour is exhibited by the Passenger Agent who needs to periodically check for matching driver agents in the system until a suitable match is found. Sequential Behaviour is used to model the Driver Agents and the Route Match Agents who need to receive a message, send a request to another agent, wait for a response from that agent and then finally send back a response to the initial calling agent. The Map Agent on the other hand can be modelled with a simple behaviour. A sequence diagram detailing how the agent behaviours interact is given below.



## 6.5 System agents



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
www.lib.mrt.ac.lk

The system consists of agents of type driver, passenger, route match agent and map agent. The flow of agent behaviour among a set of these agents is depicted in Figure 6.1. Here the initial CFP (Call For Proposals) is sent by the passenger agent. On receiving this CFP the driver agent then sends out a request to the route match agent to check if the driver and passenger routes match. The route match agent then sends out a request to the map agent to retrieve the geo location information required to run the route match algorithm. On receiving this information the route match agent runs the route match algorithm and returns the obtained results to the driver agent via an inform message. The driver agent then processes this message and sends an accept or reject response to the passenger agent.

Given below is the pseudocode for the sequential behaviour exhibited by the driver agents (DriverSequentialBehaviour) and the cyclic behavior exhibited by the passenger agents.

```

class DriverSequentialBehaviour extends SequentialBehaviour {
    DriverSequentialBehaviour(Agent a) {
        //send request to route match agent
        addSubBehaviour(new SimpleBehaviour(a) {
            boolean finished = false;
            void action() {
                String agetnName =
myAgent.getLocalName();
                startMatchAgent();
                finished = true;
            }
            public boolean done() {return finished;}
        });

        //receive response from route match agent
        addSubBehaviour(new SimpleBehaviour(a) {
            boolean finished = false;
            void action() {
                MessageTemplate mt = MessageTemplate
                .MatchPerformative(ACLMessage.INFORM);
                ACLMessage receivedMsg =
myAgent.receive(mt);
                if (receivedMsg != null) {
                    ACLMessage reply =
msgFromPassenger.createReply();
                    if
(receivedMsg.getContent().equals("NONE")) {

                        reply.setPerformative(ACLMessage.REFUSE);
                        reply.setContent("Not available");
                    }
                    else {// The requested route is available.

                        reply.setPerformative(ACLMessage.PROPOSE);

                        reply.setContent(receivedMsg.getContent());
                    }
                    myAgent.send(reply);

                    finished = true;
                } else {
                    block();
                }
            }
        }
        public boolean done() {return finished; }});}}

```



```

class PassengerRequestPerformer extends CyclicBehaviour {
    public void action() {
        switch (step) {
            case 0:
                // Send the cfp to all drivers
                ACLMessage cfp = new
ACLMessage(ACLMessage.CFP);
                for (int i = 0; i < driverAgents.length;
++i) {
                    cfp.addReceiver(driverAgents[i]);
                }
                cfp.setContent(pathPolyStr);
                cfp.setConversationId("carpool");
                cfp.setReplyWith("cfp" +
System.currentTimeMillis()); // Unique
                myAgent.send(cfp);
                // Prepare the template to get proposals
                mt = MessageTemplate.and(
MessageTemplate.MatchConversationId("carpool"),
MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
                step = 1; // Receive all proposals/refusals
from driver agents
                ACLMessage reply = myAgent.receive(mt);
                if (reply != null) {
                    // Reply received
                    if (reply.getPerformative() ==
ACLMessage.PROPOSE) {
                        saveMatchToDb(reply.getContent(),
reply.getSender().getLocalName());
                    } else {
                        //Driver refused
                    }
                    repliesCnt++;
                    if (repliesCnt >=
driverAgents.length) { // We received all replies
                        step = 2; } } else { block(); } break; } }
                public boolean done() {
                    return true;
                }
            } // End of class RequestPerformer

```



## 6.6 Route Match Agent Algorithm

The route match agent interacts with the driver and passenger agent in order to determine the route overlaps in the communication setup between a single driver and passenger agent. In order to do so, each route match agent uses the route match algorithm described below.

A match is found between a driver (person offering a ride) A and passenger B (person joining the ride) by determining whether their routes overlap. Consider as an example driver A, who is travelling from Moratuwa to Kelaniya and passenger B who needs to travel from Ratmalana to Colombo around the same time. Since B's route intersects with A's route (for travel via the Galle Road), if A was registered on the system, then B could find A on the system and setup the shared ride.

*Steps in the route match algorithm.*



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

I. Using Google Maps Directions API, encode the two routes of A and B into lists of geo location data.

II. Find the point P on A's route, which is closest to the start location of B. This will be the point where B can join A's trip. Use the haversine formula to calculate distance between points. If no point can be found within 1km proximity (threshold value), then terminate concluding routes don't overlap).

### Haversine Formula:

$$a = \sin^2(\Delta\varphi/2) + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{1-a})$$

$$d = R \cdot c$$

$\varphi$  is latitude,  $\lambda$  is longitude.

R is earth's radius (mean radius = 6,371km)

Figure 6.2:Haversine formula

III. Taking A's route and starting from point P, iterate through B's location points from start point, and calculate the distance between the corresponding location point pairs. Each location point on A that is within 1km (threshold value) proximity to B's location point is marked as a route overlap point. When no overlap is found, it is concluded that the routes do not overlap beyond this point.



University of Moratuwa, Sri Lanka  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

IV. Repeat above process for all possible alternative routes for A and B until longest overlap route, if any, is found.

The above algorithm can also be used to setup carpools by finding route overlaps from start to destination within a required threshold value (e.g. routes that overlap from start to destination with 5-10km difference).

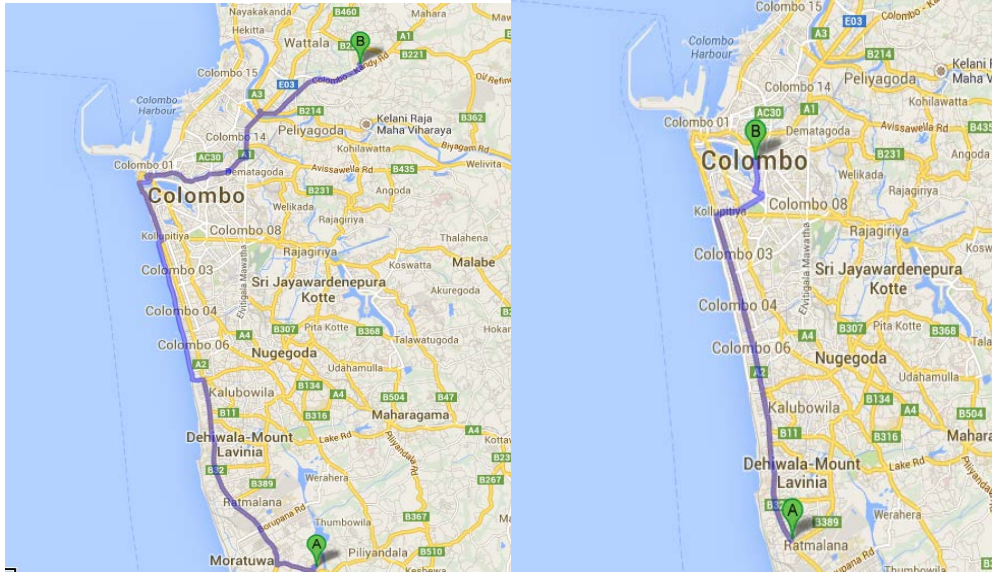


Figure 6.3: Finding journey overlap

## 6.7 Ontology layer implementation

The ontology layer of the system consists of the datasource, social and maps modules. The implementation of each of these modules is discussed next.

## 6.8 Database layer implementation

The datasource module of the system connects with a backend mysql database to read and write data. It acts as the ontology for the agents in the system. The backend database is used to store user information. It is also used to store the geospatial information extracted for user routes, especially since fetching this data using the Google Maps API is a network intensive operation.

## 6.9 Social network module

The social network module also connects with the backend system database. However it only is aware of the social connection information of the user agents. It tracks the first and second degree connections of all the agents in the system. This information is then made available to the passenger and driver agents so that they filter their search based on the social connections that they wish to maintain. For the purpose of this research social

connection data is generated by randomly pairing users in the system to establish social networks.

## 6.10 Maps data Layer

The maps module retrieves all the geospatial data required for establishing a route match. For this purpose it uses the Google Maps API which returns route information as encoded polylines. The algorithm for encoding polylines is described below. The reverse of this process is used as the polyline decoder in the route match algorithm.

Encoded polylines store two types of encoded information for any given set of points: the latitude and longitudes of those points, and the maximum zoom levels to display these points. Levels are encoded using unsigned values, while point coordinates need to use signed values, so the encoding process is slightly different for each case. This process is noted below.

The encoding process converts a binary value into a series of character codes for ASCII characters using the familiar base64 encoding scheme: to ensure proper display of these characters, encoded values are summed with 63 (the ASCII character '?') before converting them into ASCII. The algorithm also checks for additional character codes for a given point by checking the least significant bit of each byte group; if this bit is set to 1, the point is not yet fully formed and additional data must follow.

Additionally, to conserve space, **points only include the offset from the previous point** (except of course for the first point). All points are encoded in Base64 as signed integers, as latitudes and longitudes are signed values. The encoding format within a polyline needs to represent two coordinates representing latitude and longitude to a reasonable precision. Given a maximum longitude of +/- 180 degrees to a precision of 5 decimal places (180.00000 to -180.00000), this results in the need for a 32 bit signed binary integer value.

Note that the backslash is interpreted as an escape character within string literals. Any output of this utility should convert backslash characters to double-backslashes within string literals.

The steps for encoding such a signed value are specified below.

1. Take the initial signed value:

**-179.9832104**

2. Take the decimal value and multiply it by  $1e5$ , rounding the result:

**-17998321**

3. Convert the decimal value to binary. Note that a negative value must be calculated using its two's complement by inverting the binary value and adding one to the result:

**00000001 00010010 10100001 11110001**

**11111110 11101101 01011110 00001110**

**11111110 11101101 01011110 00001111**

4. Left-shift the binary value one bit:

**11111101 11011010 10111100 00011110**

5. If the original decimal value is negative, invert this encoding:

**00000010 00100101 01000011 11100001**

6. Break the binary value out into 5-bit chunks (starting from the right hand side):

**00001 00010 01010 10000 11111 00001**

7. Place the 5-bit chunks into reverse order:

**00001 11111 10000 01010 00010 00001**

8. OR each value with  $0x20$  if another bit chunk follows:

**100001 111111 110000 101010 100010 000001**

9. Convert each value to decimal:

**33 63 48 42 34 1**

10. Add 63 to each value:

**96 126 111 105 97 64**

11. Convert each value to its ASCII equivalent:

**`~oia@**



An encoded polyline also stores information specifying the precision when drawing the polyline. This information allows the map to ignore drawing segments at zoom levels where that precision is not necessary. Each point in an encoded polyline stores this information in a levels string which is also encoded alongside the encoded points.

The steps for encoding an unsigned value are specified below:

1. Take the initial unsigned value:

**174**

2. Convert the decimal value to a binary value:

**10101110**

3. Break the binary value out into 5-bit chunks (starting from the right hand side):

**101 01110**

4. Place the 5-bit chunks into reverse order:

**01110 101**

5. OR each value with 0x20 if another bit chunk follows:

**101110 00101**



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

6. Convert each value to decimal:

**46 5**

7. Add 63 to each value:

**109 68**

8. Convert each value to its ASCII equivalent:

**mD**

## **6.10 Implementing the application using JADE**

The application was implemented using JADE version 4.3. Given below is application server the system specification and the software specifications used,

- PC running Ubuntu 12.10 (32bit) with 3GB RAM
- Java version 1.7 (Open JDK)

- Java servlet version 3.1
- JADE version 4.3
- Mysql database version 5.5.31
- Mysql connector for java 5.1.3
- Apache tomcat v7 server
- Google Maps API v2
- Eclipse 4.3 IDE for development

The application is built as a hybrid of a web application and agent application. The application interface is web based and the backend operations are performed by agents running over the JADE agent platform.

### **6.11 Summary**

This section covered the system implementation. It explained the route match algorithm and the system implementation in detail using JADE. It also gave a detailed look into the JADE agent behaviours implemented by the system. The following chapter describes the system evaluation.

## Chapter 7

### Evaluation

#### 7.1 Introduction

This section describes the system evaluation process. Here we first look at the process involved in evaluating the system and look at multi agent simulation aspects. Finally a description of the system evaluation based on the multi agent simulation process is given.

#### 7.2 Evaluating the route match algorithm

The probability of a passenger finding arideshare on the system is dependent on four aspects.



University of Moratuwa, Sri Lanka.

Electronic Theses & Dissertations

www.lib.um.ac.lk

$P_t$  - The probability that the journey schedule of the passenger and driver match

$P_r$  - The probability of finding a match based on the route only

$P_s$  - The probability of a rideshare or carpool being established between two agents will depend on their social connection.

$N$  - The number of registered drivers on the system

The value of  $P_t$  will vary depending on the time of day during which the journey is scheduled. For instance, there is higher probability of finding a rideshare during rush hours (morning/evening) since there will be more people driving to/from work during these times.

The second factor is the route. If the route lies within the city in a highly urban area  $P_r$  will be higher. Also  $P_r$  will be higher for longer routes because it increases the chances of partial journey matches.

The third factor to be considered here is the social connection,  $P_s$ . This is based on whether or not the driver and passenger agents are in each other's social circle.

The fourth and final factor will be the number of registered drivers on the system. The value of  $N$  has been proven to have a deciding impact on the success of a carpooling system [19]. Based on the above factors we can depict the probability of a user finding a match on the system,  $P_{\text{match}}$  as follows,

$$P_{\text{match}} = [ 1 - ( 1 - P_t P_r P_s )^N ]$$

In a real life scenario the route match algorithm can be evaluated based on the above factors. However it is difficult to simulate the exact real life travel scenario. Therefore we look at the possibilities for simulating this scenario in a multi agent system.

### 7.3 MAS simulation



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
www.lib.mrg.ac.lk

The reasons for using simulations in MAS are twofold. (1) The deployment of the system in a real running context would have been costly and (2) real-world experiments cannot be entirely controlled so that they do not ease the development process, as they include irrelevant noise in it.

Several problems remain with these approaches considering the modeling of complex systems which involve individual entities [41]:

- Only a global perspective is possible
- Equation parameters hardly take into account the complexity of micro-level interactions
- The modeling of individual actions is not possible
- Integrating qualitative aspects is hard

There are many approaches taken towards simulating a multi agent system for the purpose of evaluation. Some of these approaches are considered next.

## 7.4 Simulation results

The system built to simulate the rideshare process was used to obtain the below results. The results are the optimal route pairings retrieved for 50 simulated users. Details of the users and their routes are presented in the Appendix C and should be referenced for a clearer understanding of the result data. The results are presented as the best match for rideshares between users and the total distance (km) covered in the rideshare.

<u>User1</u>	<u>User2</u>	<u>Rideshare Distance(km)</u>
user8	user9	26.17697656
user22	user23	13.47713173
user7	user24	13.33175753
user16	user19	10.01011473
user45	user41	9.749212024
user27	user28	8.645607179
user25	user42	7.279461059
user36	user12	6.770474731
user40	user41	6.467655213
user3	user2	6.336811231
user21	user35	6.255478447
user5	user6	4.869353604
user47	user48	4.215451781
user29	user38	4.113393987
user37	user11	3.883131685
user10	user14	2.795748377
user33	user34	2.022473248

Table 7.1: Optimal rideshares established for 50 users.

A total of 17 matches were found for the simulated users when evaluating for 50 users. The list below presents the change in number of matches found when using 10, 20, 30, 40 and 50, 60, 70, 80 and 100 users respectively for the simulation.

<u>No. of users</u>	<u>Rideshares established</u>
10	3
20	5
30	9
40	13
50	17
60	19
70	23
80	28
90	31
100	34

Table 7.2: Number of rideshares against number of simulated users.

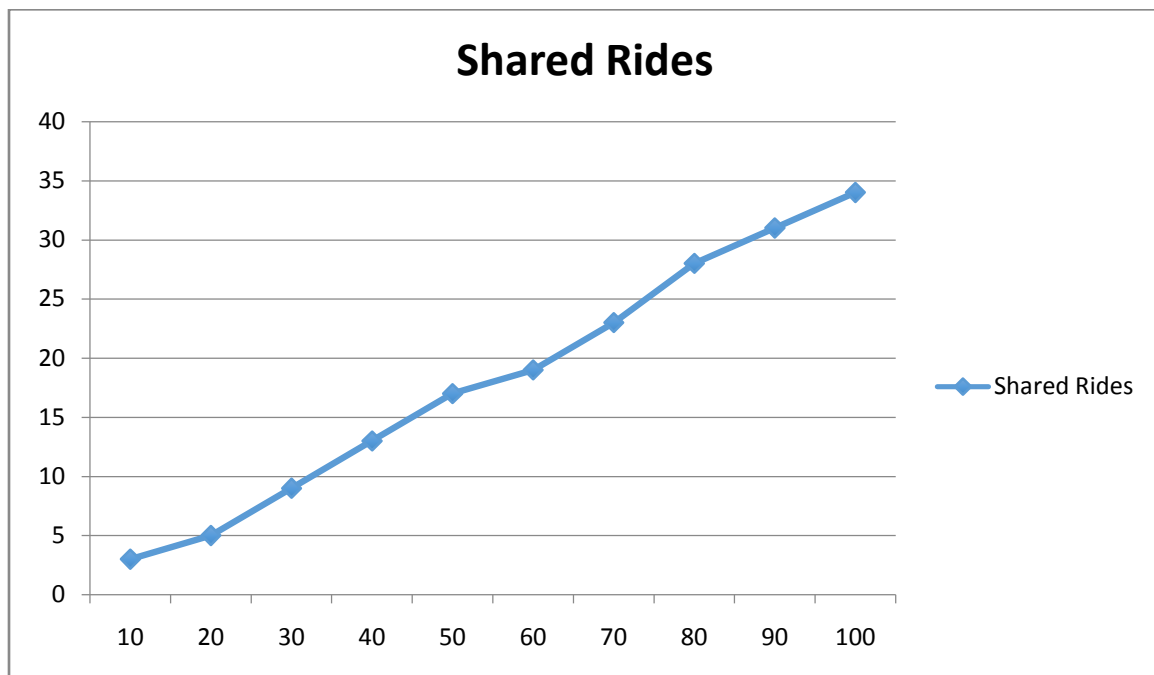


Figure 7.3: Rise in number of optimal rideshares with number of simulated users.

In order to establish carpools we look at all the possible rideshares that can be established between the user agents instead of only the optimal ones. Then a simple clustering algorithm can be applied to determine the possible carpools. A user agent pool of 100 was used in order to establish carpools since the possibility for carpools is much lower than the possibilities for rideshares because here a distance constraint is applied so that all user routes should lie within the same area. A maximum of 5km difference is allowed in order to reach acceptable results. The user agent pool is limited to 100 due to data constraints.

<b>User1</b>	<b>User2</b>	<b>Rideshare Distance(km)</b>
user8	user9	26.17697656
user22	user58	14.16150523
user22	user23	13.47713173
user7	user24	13.33175753
user22	user61	12.95906731
user16	user19	10.01011473
user7	user21	9.796667528
user27	user28	8.645607179
user74	user75	8.109333947
user26	user27	7.550970942
user26	user28	7.011995371
user78	user77	6.870268775
user67	user68	6.713441579
user67	user66	6.642564947
user68	user66	6.642564947
user3	user2	6.336811231
user21	user35	6.255478447
user75	user73	5.883826038
user4	user16	5.02370013

user43	user27	4.892435585
user43	user28	4.892435585
user6	user30	4.729175865
user70	user71	4.68626914
user55	user27	4.553465201
user55	user28	4.553465201
user26	user43	4.390089892
user47	user48	4.215451781
user26	user55	4.142660165
user50	user51	3.913976476
user16	user52	3.385574099
user1	user63	2.482907823
user20	user15	2.33398481
user46	user47	2.257422275
user1	user62	2.255909816
user33	user34	2.022473248

Table 7.3: All possible dual member carpools for 100 users.

Based on the above data the following carpools were established for the 100 user agents,

<u>User</u>	<u>Route Start</u>	<u>Route End</u>
user22	Danister de Silva Mawatha, Colombo 08	Athurugiriya RD, Homagama
user23	155, Bandaranayake Mawatha,, Colombo-12	Athurugiriya RD, Homagama
user58	Temple Road, Maradana 01000	Court Road, Homagama
user61	45, Husseiniya, Colombo 12	Athurugiriya RD, Homagama

Table 7.4: Carpool found with 4 users.



<u>User</u>	<u>Route Start</u>	<u>Route End</u>
user26	Auburn Place, Dehiwala	Kynsey road, Colombo 08
user27	Templer's Road, Mount Lavinia	Kynsey road, Colombo 08
user28	Templer's Road, Mount Lavinia	62, Gregory's Road,, Colombo 07

Table 7.5: Carpool found with 3 users.

<u>User</u>	<u>Route Start</u>	<u>Route End</u>
user66	Court Road, Homagama	Ministry of Education, Pelawatta
user67	Athurugiriya RD, Homagama 10200	Ministry of Education, Pelawatta
user68	habarakada homagama	Ministry of Education, Pelawatta

Table 7.6: Carpool found with 3 users.

The above data clearly shows the need for a large number of users in order to successfully establish carpools of size  $> 2$ .

## 7.5 Summary

This section covered the system evaluation. It presented an evaluation of the route match algorithm and then presented various methods for evaluating a multi agent based system using simulation. Finally it presented the multi agent features of the system and the results from the system simulation. The next chapter presents the research conclusion.

## Conclusion

### 8.1 Introduction

This chapter follows the project evaluation and presents the final outcome of this research project. It discusses the issues that were faced during the carrying out of this research and finally lists the opportunities available to expand on this research.

### 8.2 Conclusion

When evaluating the success of the project what must be considered is the objectives that were set during the onset of the project. The objectives that were initially listed in the project proposal are given below for reference,

Project Objectives:  University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

- Provide a system that gives a car pooling solution for registered users
- Provide a system that provides ride sharing option
- Provide the system users with best solutions based on their route requirements, time requirement and personal preferences.
- Provide a safe and social car pooling/ride sharing system
- Provide a user friendly and personalized solution

The primary objectives were to build a system that would provide an effective solution to the car pooling and ride sharing problem. The solution provided was also expected to be personalized and was expected to exactly fit the time and route requirements of the user who will act as the passenger. When considering the outcome of the project it can be concluded that the initial project requirements have been met successfully. However in terms of user experience the system built has much room for improvement. Although not

listed above one of the main requirements of this project was to successfully complete a research on the use of a multi agent system to solve the joint ride sharing and car pooling problem. In that sense the project has been a success.

When considering the requirement for social connectivity the project does not provide a full solution since as a research project connecting to a real live social platform will result in unnecessary noise and distraction to the initial research. Therefore instead attempts were made to simulate social connection amongst the system users and to use these simulated connections to evaluate the impact on the emergent properties of the multi agent system.

Many difficulties arose during the implementation of the project. The main issue that was faced was the requirement to connect the system web interface with the JADE agent container. The solution that I used to solve the problem was to implement the web interface as a JAVA servlet in separate package inside the JADE project. The servlet code could then call the JADE agent code.



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

With respect to the system web interface there was also the requirement of displaying the route match results to the user. Since the system runs the route match process asynchronously and because route match results vary over time (depending on changes to the system and the environment) I decided to use a separate page to list the route match results. Initially it was planned to implement a mobile based notification system to notify the users of the route match results. However due to the complexity involved in integrating the web interface with the JADE container I later decided not to implement the notification module. Therefore the notification module will be listed under future work for this project.

Yet another difficulty faced during the implementation was the requirement to connect the system to a persistent database for data storage and retrieval. This was achieved by utilizing a java mysql connector package and a dedicated DatabseAgent who will handle all connections to the database and respond to datastore resource requests coming from other agents.

Building the route matching algorithm for the system also took considerable time and effort especially since although the past literature provided numerous algorithmic and non algorithmic ways for solving the car pooling and ride sharing problem, non of these solutions dealt in detail with solving the overlapping route requirement.

Implementing the agent behaviors for the system also proved to be a tough challenge especially since there were multiple asynchronous messaging process that in certain cases needed to be aligned so that requests and responses would flow in the correct order. I solved these requirements by implementing different JADE agent behaviour models for the agents as per their requirements.

Another main concern during the project implementation was the source for reliable geographic data. I selected Google Maps API as the source for geographic data since it has an open API that supports numerous functions including alternative route retrieval, reverse geocoding and distance matrix calculation.



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

Finally it can be concluded that the project was successful to a level that at least three quarters of its objectives were achieved. Work that was not successful will be considered as future work and listed below.

### **8.3 Future work**

Much of the focus in this research was on the implementation of the multi-agent based solution for carpooling and the journey matching algorithm. There also are many possibilities for improving the solution based on personal and social preferences. Further improvements to this research would lie in this area and would describe the role that social and cultural connections play in the implementation of a carpooling solution. Also the project evaluation was conducted by utilizing rather small scopes of agents (in the order of tens). A more detailed analysis could be carried out by using more complex scopes and variables.

## 8.4 Summary

The chapter concludes this research project by presenting a discussion on the overall outcomes of this project. It ties together the research description given in the literature review, technology, approach, design, implementation and evaluation chapters. It also outlines the various issues that were faced during the implementation of the project. In conclusion it also lists the future work for this research.



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

## References

- [1] Srivastava, B. (2012). Making Car Pooling Work—Myths and Where to Start. In *19th ITS World Congress*.
- [2] Graziotin, D. (2013). An Analysis of issues against the adoption of Dynamic Carpooling. *arXiv preprint arXiv:1306.0361*.
- [3] Why carpooling will never work in India - YourStory.com. (2013, May). Retrieved January 25, 2014, from <http://yourstory.com/2013/05/carpooling-will-fail-in-india/>
- [4] *High fuel prices: Is car-pooling an option?* (2011, December 11). Retrieved from <http://www.sundaytimes.lk/111211/BusinessTimes/bt09.html>
- [5] Kamar, E., & Horvitz, E. (2009, July). Collaboration and Shared Plans in the Open World: Studies of Ridesharing. In *IJCAI* (Vol. 9, p. 187).
- [6] Kamar, E., Horvitz, E., & Meek, C. (2008, May). Mobile opportunistic commerce: mechanisms, architecture, and application. In Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems, Volume 2 (pp. 1087-1094). International Foundation for Autonomous Agents and Multiagent Systems.
- [7] Horvitz, E., Koch, P., Kadie, C. M., & Jacobs, A. (2002, August). Coordinate: Probabilistic forecasting of presence and availability. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence* (pp. 224-233). Morgan Kaufmann Publishers Inc..
- [8] Horvitz, E., Koch, P., Sarin, R., Apacible, J., & Subramani, M. (2005). Bayesphone: Precomputation of context-sensitive policies for inquiry and action in mobile devices. In *User Modeling 2005* (pp. 251-260). Springer Berlin Heidelberg.
- [9] Krumm, J., & Horvitz, E. (2005). The Microsoft multiperson location survey. Microsoft Research Technical Report.
- [10] Di Martino, S., Galiero, R., Giorio, C., Ferrucci, F., & Sarro, F. (2011). A Matching-Algorithm based on the Cloud and Positioning Systems to Improve Carpooling. In *DMS* (pp. 90-95).
- [11] Kothari, A. B. (2004). Genghis-a multiagent carpooling system. *Bath: Department of Computer Science, University of Bath*.

- [12] Knapen, L., Keren, D., Yasar, A. U. H., Cho, S., Bellemans, T., Janssens, D., & Wets, G. (2012). Analysis of the co-routing problem in agent-based carpooling simulation. *Procedia Computer Science*, 10, 821-826.
- [13] Cho, S., Yasar, A., & Knapen, L. (2011). CarPoolingAgent-based carpooling design (Part-A). IMOB internal report.
- [14] Gallo, G., Longo, G., Pallottino, S., & Nguyen, S. (1993). Directed hypergraphs and applications. *Discrete applied mathematics*, 42(2), 177-201.
- [15] ‘Cubillos, C., & Guidi-Polanco, F. (2006). An Agent Solution to Flexible Planning and Scheduling of Passenger Trips. In *Artificial Intelligence in Theory and Practice* (pp. 355-364). Springer US.’
- [16] Bellifemine, F. L., Caire, G., & Greenwood, D. (2007). *Developing multi-agent systems with JADE* (Vol. 7). John Wiley & Sons.
- [17] Cubillos, C., Guidi-Polanco, F., & Demartini, C. (2004, October). Multi-agent infrastructure for distributed planning of demand-responsive passenger transportation service. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on* (Vol. 2, pp. 2013-2017). IEEE.
- [18] Manzini, R., & Pareschi, A. (2012). A decision-support system for the car pooling problem. *Journal of Transportation Technologies*, 2, 85.
- [19] Morris, J. (2008). *Saferide: Reducing single occupancy vehicles*. Technical report.
- [20] Knapen, L., Yasar, A., Cho, S., Keren, D., Dbai, A. A., Bellemans, T., ...& Bhaduri, K. (2013). Exploiting graph-theoretic tools for matching in carpooling applications. *Journal of Ambient Intelligence and Humanized Computing*, 1-15.
- [21] Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., & Edwards, D. D. (1995). *Artificial intelligence: a modern approach* (Vol. 2). Englewood Cliffs: Prentice hall.
- [22] FIPA, Foundation for Intelligent Physical Agents, website: <http://www.fipa.org>
- [23] Brooks, R. A. (1986). A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1), 14-23.
- [24] Rao, A.S. and Georgeff, M. BDI Agents: from Theory to Practice. In *Proceedings of the 1st International Conference on Multi-Agent Systems*, pp. 312–319, San Francisco, CA, 1995.

- [25] Georgeff, M. and Lansky, A. Reactive Reasoning and Planning: an Experiment with a Mobile Robot. In Proceedings of the 7th National Conference on Artificial Intelligence, pp. 677–682, Seattle, WA, 1987.
- [26] Ferguson, I.A. Towards an Architecture for Adaptive, Rational, Mobile Agents. In Werner, E. and Demazeau, Y. (eds), Decentralized AI 3 – Proceedings of the Third European Workshop on Modelling Autonomous Agents and Multi-Agent World, pp. 249–262, Elsevier, Amsterdam, The Netherlands, 1991.
- [27] Muller, J.P., Pischel, M. and Thiel, M. Modelling Reactive Behaviour in Vertically Layered Agent Architectures. In Wooldridge, M. and Jennings, N.R. (eds), Intelligent Agents: Theories, Architectures, and Languages (LNAI 890), pp. 261–276, Springer-Verlag, Heidelberg, 1995.
- [28] Searle, J. Speech Acts, Cambridge, MA, Cambridge University Press, 1969.
- [29] Mayfield, J., Labrou, Y. and Finin, T. Evaluating KQML as an Agent Communication Language. In Wooldridge, M., Muller, J.P. and Tambe, M. (eds), Intelligent Agents II (LNAI 1037), pp. 347–360. Springer-Verlag, Heidelberg, 1996.
- [30] Genesereth, M.R. and Ketchpel, S.P. Software Agents, Communications of the ACM, 37(7): pp. 48–53, 1994.
- [31] Labrou, Y., Finin T., and Peng, Y. Agent Communication Languages: the Current Landscape. IEEE Intelligent Systems, 14(2): pp. 45–52, 1999.
- [32] Nwana, H.S., Lee, L. and Jennings, N.R. Coordination in Software Agent Systems. BT Technology Journal, 14(4): pp. 79–88, 1996.
- [33] Durfee, E. Distributed Problem Solving and Planning. In Weiß, Gerhard, (ed.), Multiagent Systems: a Modern Approach to Distributed Artificial Intelligence, pp. 121–164, MIT Press, Cambridge, MA, 1999.
- [34] Smith, R. and Davis, R. The Contract Net protocol: High Level Communication and Control in a Distributed Problem Solver. IEEE Transactions on Computers, 29(12): pp. 1104–1113, 1980.
- [35] Georgeff, M., Communication and Interaction in Multi-Agent Planning. In Proceedings of the 3rd National Conference on Artificial Intelligence, pp. 125–129, Washington, DC, 1983.
- [36] Georgeff, M., A Theory of Action for Multi Agent Planning, Proceedings of the 4th National Conference on Artificial Intelligence, pp. 121–125, Austin, TX, 1984.



- [37] Durfee, E. and Victor, L. Using Partial Global Plans to Coordinate Distributed Problem Solvers. In Proceedings of the 10th International Joint Conference on Artificial Intelligence, pp. 875–883, Milan, August 1987.
- [38] Bussmann, S. and Muller, J. A Negotiation Framework for Co-operating Agents. In Deen, S.M. (ed.), Proceedings of CKBS-SIG, pp. 1–17, Keele, UK, 1992.
- [39] Garson G. Quantification in Modal Logic. In Handbook of Philosophical Logic, Vol. II: Extensions of Classical Logic, pp. 249–307, D. Reidel Publishing Company, 1984.
- [40] B. Burmeister, A. Haddadi, G. Matylis, Application of multi-agent systems in traffic and transportation, IEE Proceedings on Software Engineering 144 (1) (1997) 51–60
- [41] Ferber, J. (1999). Multi-agent systems: an introduction to distributed artificial intelligence (Vol. 1). Reading: Addison-Wesley.



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

# Appendix A:

## JADE Architecture Overview

### A.1 Introduction

This appendix gives an architectural overview of the JADE platform as retrieved from the <http://jade.tilab.com/> website.

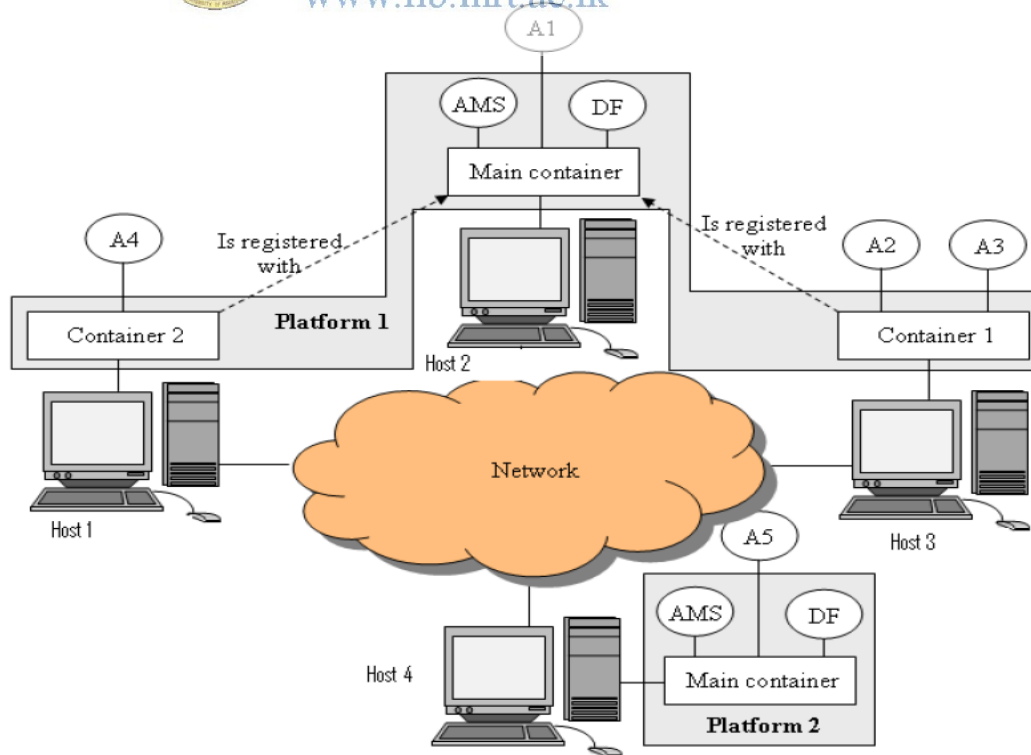
### A.2 JADE Architecture Overview

This provides an overview of the JADE Architecture introducing the notions of

- Agent
- Container
- Platform
- Main Container
- AMS and DF



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)



The figure represents the main JADE architectural elements. An application based on JADE is made of a set of components called *Agents* each one having a unique name. Agents execute tasks and interact by exchanging messages. Agents live on top of a *Platform* that provides them with basic services such as message delivery. A platform is composed of one or more *Containers*. Containers can be executed on different hosts thus achieving a distributed platform. Each container can contain zero or more agents.

For instance, with reference to the picture, container "Container 1" in host *Host 3* contains agents A2 and A3. Even if in some particular scenarios this is not always the case, you can think of a Container as a JVM (so, 1 JVM ==> 1 container ==> 0 or many agents). A special container called *Main Container* exists in the platform.

The main container is itself a container and can therefore contain agents, but differs from other containers as

1. It must be the first container to start in the platform and all other containers register to it at bootstrap time
2. It includes two special agents: the *AMS* that represents the authority in the platform and is the only agent able to perform platform management actions such as starting and killing agents or shutting down the whole platform (normal agents can request such actions to the AMS). The *DF* that provides the Yellow Pages service where agents can publish the services they provide and find other agents providing the services they need.

It should be noticed that if another main container is started, as in host *Host 4*, this constitutes a new platform.

### Agent communication

Agents can communicate transparently regardless of whether they live in the same container (e.g. A2 and A3), in different containers (in the same or in different hosts) belonging to the same platform (e.g. A1 and A2) or in different platforms (e.g. A1 and A5). Communication is based on an asynchronous message passing paradigm.

Message format is defined by the ACL language defined by FIPA, an international organization that issued a set of specifications for agent interoperability. An ACL Message contains a number of fields including

- the sender
- the receiver(s)
- the communicative act (also called performative) that represents the intention of the sender of the message. For instance when an agent sends an INFORM message it wishes the receiver(s) to become aware about a fact (e.g. (INFORM "today it's raining")). When an agent sends a REQUEST message it wishes the receiver(s) to perform an action. FIPA defined 22 communicative acts, each one with a well defined semantics, that ACL gurus assert can cover more than 95% of all possible situations. Fortunately in 99% of the cases we don't need to care about the formal semantics behind Communicative acts and we just use them for their intuitive meaning.
- the content i.e. the actual information conveyed by the message (the fact the receiver should become aware of in case of an INFORM message, the action that the receiver is expected to perform in case of a REQUEST message)



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
www.lib.moratuwa.lk

## Appendix B:

### List of simulated users and their routes

#### B.1 Introduction

This appendix gives the list of users and their routes that were simulated to evaluate the system functionality.

#### B.2 List of addresses used to simulate user routes

The below address list taken from a listing of schools in the Western Province of Sri Lanka was used to generate the route list for simulated users. This list was chosen because it gives a well-distributed list of addresses in the Western Province of Sri Lanka. Therefore it can be effectively used to simulate a rideshare system in the Western Province.



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations

1. Maligakanda road, Colombo 10 01000
2. PADOGA ROAD, KOTTE 10100
3. AdikaramMawatha, Kotte 10010
4. Baseline Road, Dematagoda, Colombo-9 00900
5. wp/ja/mahamathayvidyalaya, Athurugiriya 12010
6. Hokandara South, Hokandara 10118
7. AnandaRajakarunaMawatha, Colombo 10 01000
8. Kularathnamawatha, Colombo 10 0015
9. High Level Rd., Nugegoda, 01
10. Katubedda, Moratuwa
11. Bope, Padukka
12. Buddhagosha M.V., Kalubowila 12056
13. Highlevel Road, Maharagama 10400
14. Bomiriya National School, Colombo
15. Boralessgamuwa M.V, Boralessgamuwa 10290
16. Station Road, Mount Lavinia
17. DanisterdeSilvaMawatha, Colombo 08 00025

18. 155, BandaranayakeMawatha, Colombo-12 0094
19. Auburn Place, Dehiwala
20. Templer\'s Road, Mount Lavinia 0094
21. Godagama Road, Athurugiriya 90018
22. ., Piliyandala 10129
23. Kynsey road, Colombo 08
24. 62, Gregory\'s Road, Colombo 07
25. Moratuwa, Moratuwa
26. Brahmanagama, Pannipitiya
27. No 34, Mallay Street, Colombo 02
28. DarmapalaMawatha, Dehiwala 011
29. SriJayawardhanapuraMawatha, Borella 00800
30. Kesbewa, Piliyandala
31. Diyagama, Kiriwattuduwa. 10208
32. Park Road, Colombo 05
33. DharmapalaVidyalaya, Pannipitiya
34. Hotel Road, Mount Lavinia
35. wasala road, colombo 13
36. Temple Road, Maradana 01000
37. habarakada, homagama
38. 45, Husseiniya, Colombo 12 3012
39. Dam Street, Colombo 12
40. Court Road, Homagama
41. Athurugiriya RD, Homagama 10200
42. Salamulla, Kolonnawa 10600
43. 207/1, DharmapalaMawatha, Colombo 7 00700
44. Ministry of Education, Pelawatta
45. Ministry of Education, Pelawatta
46. Jalthara. Hanwella, Hanwella 1224
47. School Lane, Nawala, Rajagiriya, Colombo 011
48. 166, Dematagoda Road,, Colombo 09 011
49. Kosgama
50. kosgama, kosgama 00255
51. MahaVidyalayaMawatha, Colombo 13 01300

52. Hokandara Road, Pannipitiya 10230
53. Mulleriyawa New town
54. Kandawala Road, Ratmalana
55. Horana Road, Kottawa, Pannioitiya
56. Hena Road, Mount Lavinia 10370
57. Havelock Town, Colombo 05
58. No. 724, Galle Road, Colombo 03
59. Thalangama North, Bathtaramulla 10120
60. Madiwela, Kotte
61. magamma, Homagama
62. Foster Lane, Colombo
63. Pepiliyana Road, Nugegoda 24250
64. Gammana Road, Maharagama
65. Bokundara, Piliyandala
66. Makuluduwa, Piliyandala
67. New Kandy Road, Malabe 094
68. Horana Road, Mattegoda, Pannipitiya.
69. Mawathgama, Homagama 10220
70. Mayadunna M.V., Hanwella
71. Padukka Road, Meegoda 10504
72. Meegoda, 10504
73. Meethotamulla Road, Kolonnawa
74. Kensington gardens, Colombo 04.,, Colombo 0004
75. New Kandy Rd., Malabe
76. Siridammamawatha, Colombo
77. Hiripitiya, Pannipitiya 10230
78. stanleythilakaratna mw, nugegoda
79. High Level Road, Maharagama
80. Isurupaya, Battaramulla
81. SiriPiyararhana Central College, Padukka
82. Pahathgama, Hanwella
83. Madapatha, Piliyandala
84. Pinnawala , Waga, Padukka
85. WP/HO/Pitipanam.v. Pitpana North, Homagama

### B.3 List of simulated users and their routes

Using the above address list in a random pairing algorithm the below user route list was generated. This generated data was used to simulate a rideshare/carpool system. Results of the simulation were presented in the evaluation chapter.

User	Route Start	Route End
1	Maligakanda road,, Colombo 10 01000	Baseline Road, Dematagoda,Colombo-9 00900
2	PADOGA ROAD, KOTTE 10100	Hokandara South, Hokandara 10118
3	AdikaramMawatha, Kotte 10010	Hokandara South, Hokandara 10118
4	AdikaramMawatha, Kotte 10010	AnandaRajakarunaMawatha, Colombo 10 01000
5	Hokandara South, Hokandara 10118	High Level Rd., Nugegoda, 01
6	Hokandara South, Hokandara 10118	Katubedda, Moratuwa
7	AnandaRajakarunaMawatha, Colombo 10 01000	Katubedda, Moratuwa
8	AnandaRajakarunaMawatha, Colombo 10 01000	Pinnawala , Waga, Padukka
9	Kularathnamawatha, Colombo 10 0015	Pinnawala , Waga, Padukka
10	High Level Rd., Nugegoda, 01	High Level Road, Maharagama
11	Katubedda, Moratuwa	High Level Road, Maharagama
12	Katubedda, Moratuwa	Siridammamawatha, Colombo
13	Pinnawala , Waga, Padukka	Siridammamawatha, Colombo
14	Pinnawala , Waga, Padukka	Boralessgamuwa M.V, Boralessgamuwa 10290
15	High Level Road, Maharagama	Station Road, Mount Lavinia
16	High Level Road, Maharagama	Danister de Silva Mawatha, Colombo 08 00025
17	Siridammamawatha, Colombo	Danister de Silva Mawatha, Colombo 08 00025
18	Siridammamawatha, Colombo	155,Bandaranayeke Mawatha,, Colombo-12 0094
19	Boralessgamuwa M.V, Boralessgamuwa 10290	155,Bandaranayeke Mawatha,, Colombo-12 0094
20	Boralessgamuwa M.V, Boralessgamuwa 10290	Auburn Place, Dehiwala
21	Danister de Silva Mawatha, Colombo 08 00025	Templer\'s Road, Mount Lavinia 0094
22	Danister de Silva Mawatha, Colombo 08 00025	Athurugiriya RD, Homagama 10200
23	155,Bandaranayeke Mawatha,, Colombo-12 0094	Athurugiriya RD, Homagama 10200
24	155,Bandaranayeke Mawatha,, Colombo-12 0094	., Piliyandala 10129
25	Auburn Place, Dehiwala	., Piliyandala 10129
26	Auburn Place, Dehiwala	Kynsey road, Colombo 08
27	Templer\'s Road, Mount Lavinia 0094	Kynsey road, Colombo 08
28	Templer\'s Road, Mount Lavinia 0094	62, Gregory\'s Road,, Colombo 07



29	Athurugiriya RD, Homagama 10200	62, Gregory\'s Road,, Colombo 07
30	Athurugiriya RD, Homagama 10200	Moratuwa, Moratuwa
31	., Piliyandala 10129	Brahmanagama, Pannipitiya
32	Kynsey road, Colombo 08	Brahmanagama, Pannipitiya
33	Kynsey road, Colombo 08	No 34, Mallay Street, Colombo 02
34	62, Gregory\'s Road,, Colombo 07	No 34, Mallay Street, Colombo 02
35	62, Gregory\'s Road,, Colombo 07	DarmapalaMawatha, Dehiwala 011
36	Moratuwa, Moratuwa	DarmapalaMawatha, Dehiwala 011
37	Moratuwa, Moratuwa	Sri JayawardhanapuraMawatha, Borella 00800
38	Brahmanagama, Pannipitiya	Sri JayawardhanapuraMawatha, Borella 00800
39	Brahmanagama, Pannipitiya	Kesbewa, Piliyandala
40	No 34, Mallay Street, Colombo 02	Kesbewa, Piliyandala
41	No 34, Mallay Street, Colombo 02	Diyagama, Kiriwattuduwa. 10208
42	DarmapalaMawatha, Dehiwala 011	Diyagama, Kiriwattuduwa. 10208
43	DarmapalaMawatha, Dehiwala 011	Park Road, Colombo 05
44	Sri JayawardhanapuraMawatha, Borella 00800	Park Road, Colombo 05
45	Sri JayawardhanapuraMawatha, Borella 00800	DharmapalaVidyalaya, Pannipitiya
46	Kesbewa, Piliyandala	DharmapalaVidyalaya, Pannipitiya
47	Kesbewa, Piliyandala	Hotel Road, Mount Lavinia
48	Diyagama, Kiriwattuduwa. 10208	Hotel Road, Mount Lavinia
49	Diyagama, Kiriwattuduwa. 10208	wasala road, colombo 13
50	Park Road, Colombo 05	wasala road, colombo 13
51	Park Road, Colombo 05	Temple Road, Maradana 01000
52	DharmapalaVidyalaya, Pannipitiya	Temple Road, Maradana 01000
53	DharmapalaVidyalaya, Pannipitiya	habarakada, homagama
54	Hotel Road, Mount Lavinia	habarakada, homagama
55	Hotel Road, Mount Lavinia	45, Husseiniya, Colombo 12 3012
56	wasala road, colombo 13	Dam Street, Colombo 12
57	Temple Road, Maradana 01000	Dam Street, Colombo 12
58	Temple Road, Maradana 01000	Court Road, Homagama
59	habarakada, homagama	Court Road, Homagama
60	habarakada, homagama	Athurugiriya RD, Homagama 10200
61	45, Husseiniya, Colombo 12 3012	Athurugiriya RD, Homagama 10200
62	45, Husseiniya, Colombo 12 3012	Meethotamulla Road, Kolonnawa
63	Dam Street, Colombo 12	Meethotamulla Road, Kolonnawa
64	Dam Street, Colombo 12	207/1, DharmapalaMawatha, Colombo 7
65	Court Road, Homagama	00700
		207/1, DharmapalaMawatha, Colombo 7

	00700
66 Court Road, Homagama	Ministry of Education, Pelawatta
67 Athurugiriya RD, Homagama 10200	Ministry of Education, Pelawatta
68 habarakada, homagama	Ministry of Education, Pelawatta
69 Meethotamulla Road, Kolonnawa	Ministry of Education, Pelawatta
70 Meethotamulla Road, Kolonnawa	Jalthara. Hanwella, Hanwella 1224
71 207/1, DharmapalaMawatha, Colombo 7 00700	Jalthara. Hanwella, Hanwella 1224
	School Lane, Nawala,Rajagiriya, Colombo
72 207/1, DharmapalaMawatha, Colombo 7 00700	011
	School Lane, Nawala,Rajagiriya, Colombo
73 Ministry of Education, Pelawatta	011
74 Ministry of Education, Pelawatta	166, Dematagoda Road,, Colombo 09 011
75 Ministry of Education, Pelawatta	166, Dematagoda Road,, Colombo 09 011
76 Ministry of Education, Pelawatta	Kosgama
77 Jalthara. Hanwella, Hanwella 1224	Kosgama
78 Jalthara. Hanwella, Hanwella 1224	kosgama, kosgama 00255
79 School Lane, Nawala,Rajagiriya, Colombo 011	kosgama, kosgama 00255
80 School Lane, Nawala,Rajagiriya, Colombo 011	MahaVidyalamawatha, Colombo 13 01300
81 166, Dematagoda Road,, Colombo 09 011	MahaVidyalamawatha, Colombo 13 01300
82 166, Dematagoda Road,, Colombo 09 011	Hokandara Road, Pannipitiya 10230
83 Kosgama	Hokandara Road, Pannipitiya 10230
84 Kosgama	Mulleriyawa New town
85 kosgama, kosgama 00255	Mulleriyawa New town
86 kosgama, kosgama 00255	Kandawala Road, Ratmalana
87 MahaVidyalamawatha, Colombo 13 01300	Kandawala Road, Ratmalana
88 MahaVidyalamawatha, Colombo 13 01300	Horana Road, Kottawa, Pannioitiya
89 Hokandara Road, Pannipitiya 10230	Horana Road, Kottawa, Pannioitiya
90 Hokandara Road, Pannipitiya 10230	Hena Road, Mount Lavinia 10370
91 Mulleriyawa New town	Hena Road, Mount Lavinia 10370
92 Mulleriyawa New town	Havelock Town, Colombo 05
93 Kandawala Road, Ratmalana	Havelock Town, Colombo 05
94 Kandawala Road, Ratmalana	No. 724, Galle Road, Colombo 03
95 Horana Road, Kottawa, Pannioitiya	No. 724, Galle Road, Colombo 03
96 Horana Road, Kottawa, Pannioitiya	Thalangama North, Bathtaramulla 10120
97 Hena Road, Mount Lavinia 10370	Thalangama North, Bathtaramulla 10120
98 Hena Road, Mount Lavinia 10370	Madiwela, Kotte
99 Havelock Town, Colombo 05	Madiwela, Kotte
100 Havelock Town, Colombo 05	magamma, Homagama

## Appendix C:

### FIPA ACL Message Structure

#### C.1 Introduction

This appendix gives a detailed description of the FIPA ACL message structure taken from the FIPA standard specification SC00061.

#### C.2 FIPA ACL Message Structure

A FIPA ACL message contains a set of one or more message parameters. Precisely which parameters are needed for effective agent communication will vary according to the situation; the only parameter that is mandatory in all ACL messages is the performative, although it is expected that most ACL messages will also contain sender, receiver and content parameters.



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
www.lib.mrt.ac.lk

If an agent does not recognize or is unable to process one or more of the parameters or parameter values, it can reply with the appropriate not-understood message.

Specific implementations are free to include user-defined message parameters other than the FIPA ACL message parameters specified in *Table 1*. The semantics of these user-defined parameters is not defined by FIPA, and FIPA compliance does not require any particular interpretation of these parameters. The prefatory string “X-” must be used for the names of these non-FIPA standard additional parameters.

Some parameters of the message might be omitted when their value can be deduced by the context of the conversation. However, FIPA does not specify any mechanism to handle such conditions, therefore those implementations that omit some message parameters are not guaranteed to interoperate with each other.

The full set of FIPA ACL message parameters is shown in *Table 1* without regard to their specific encodings in an implementation. FIPA-approved encodings and parameter orderings for ACL messages are given in other specifications. Each ACL message representation specification contains precise syntax descriptions for ACL message encodings based on XML, text strings and several other schemes.

A FIPA ACL message corresponds to the abstract parameter message payload identified in the [FIPA00001].

<b>Parameter</b>	<b>Category of Parameters</b>
performative	Type of communicative acts
sender	Participant in communication
receiver	Participant in communication
reply-to	Participant in communication
content	Content of message
language	Description of Content
encoding	Description of Content
ontology	Description of Content
protocol	Control of conversation
conversation-id	Control of conversation
reply-with	Control of conversation
in-reply-to	Control of conversation
reply-by	Control of conversation

The following terms are used to define the ontology and the abstract syntax of the FIPA ACL message structure:

- **Frame.** This is the mandatory name of this entity that must be used to represent each instance of this class.
- **Ontology.** This is the name of the ontology, whose domain of discourse includes their parameters described in the table.
- **Parameter.** This identifies each component within the frame. The type of the parameter is defined relative to a particular encoding. Encoding specifications for ACL messages are given in their respective specifications.
- **Description.** This is a natural language description of the semantics of each parameter. Notes are included to clarify typical usage.
- **Reserved Values.** This is a list of FIPA-defined constants associated with each parameter. This list is typically defined in the specification referenced.



University of Moratuwa, Sri Lanka.  
 Electronic Theses & Dissertations  
[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

All of the FIPA message parameters share the frame and ontology shown in *Table 2*.

<b>Frame</b>	fipa-acl-message
<b>Ontology</b>	fipa-acl

## Type of Communicative Act

### Performative

Parameter	Description	Reserved Values
performative	Denotes the type of the communicative act of the ACL message	See [FIPA00037]

**Notes:** The performative parameter is a required parameter of all ACL messages. Developers are encouraged to use the FIPA standard performatives (see [FIPA00037]) whenever possible.

## Participants in Communication

### Sender

Parameter	Description	Reserved Values
sender	Denotes the identity of the sender of the message, that is, the name of the agent of the communicative act.	

**Notes:** The sender parameter will be a parameter of most ACL messages. It is possible to omit the sender parameter if, for example, the agent sending the ACL message wishes to remain anonymous. The sender parameter refers to the agent which performs the communicative act giving rise to this ACL message.

### Receiver



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations

[www.lib.mrt.ac.lk](http://www.lib.mrt.ac.lk)

Parameter	Description	Reserved Values
receiver	Denotes the identity of the intended recipients of the message.	

**Notes:** Ordinarily, the receiver parameter will be a part of every ACL message. It is only permissible to omit the receiver parameter if the message recipient can be reliably inferred from context, or in special cases such as the embedded ACL message in proxy and propagate.


The receiver parameter may be a single agent name or a non-empty set of agent names. The latter corresponds to the situation where the message is multicast. Pragmatically, the semantics of this multicast is that the sender intends the message for each recipient of the CA encoded in the message. For example, if an agent performs an inform act with a set of three agents as receiver, it denotes that the sender intends each of these agents to come to believe the content of the message.

## Reply To

Parameter	Description	Reserved Values
reply-to	This parameter indicates that subsequent messages in this conversation thread are to be directed to the agent named in thereply-to parameter, instead of to the agent named in the senderparameter.	

## Content of Message

### Content

Parameter	Description	Reserved Values
content	 Denotes the content of the message. Equivalently denotes the object of the action. The meaning of the content of any ACL message is intended to be interpreted by the receiver of the message. This is particularly relevant for instance when referring to referential expressions, whose interpretation might be different for the sender and the receiver. <i>University of Moratuwa, Sri Lanka. Electronic Theses &amp; Dissertations www.lib.mrt.ac.lk</i>	

**Notes:** Most ACL messages require a content expression. Certain ACL message types, such as cancel, have an implicit content, especially in cases of using the conversation-id or in-reply-to parameters.

## Description of Content

### Language

Parameter	Description	Reserved Values
language	Denotes the language in which the content parameter is expressed.	See [FIPA00007]

**Notes:** The ACL content parameter is expressed in a formal language. This field may be omitted if the agent receiving the message can be assumed to know the language of the content expression.

### Encoding

Parameter	Description	Reserved Values
encoding	Denotes the specific encoding of the content language expression.	See [FIPA00007]

**Notes:** The content expression might be encoded in several ways. The encoding parameter is optionally used to specify this encoding to the recipient agent. If the encoding parameter is not present, the encoding will be specified in the message envelope that encloses the ACL message.

### Ontology

Parameter	Description	Reserved Values
ontology	Denotes the ontology(s) used to give a meaning to the symbols in the content expression.	

**Notes:** The ontology parameter is used in conjunction with the language parameter to support the interpretation of the content expression by the receiving agent. In many



situations, the ontology parameter will be commonly understood by the agent community and so this message parameter may be omitted.

## Control of Conversation

### Protocol

Parameter	Description	Reserved Values
protocol	Denotes the interaction protocol that the sending agent is employing with this ACL message.	See [FIPA00025]

**Notes:** The protocol parameter defines the interaction protocol in which the ACL message is generated. This parameter is optional; however, developers are advised that employing ACL without the framework of an interaction protocol (and thus directly using the ACL semantics to control the agent's generation and interpretation of ACL messages) is an extremely ambitious undertaking.



University of Moratuwa, Sri Lanka.  
Electronic Theses & Dissertations  
www.lib.mrt.ac.lk

Any ACL message that contains a non-null value for the protocol parameter is considered to belong to a conversation and it is required to respect the following rules:

- the initiator of the protocol must assign a non-null value to the conversation-id parameter,
- all responses to the message, within the scope of the same interaction protocol, should contain the same value for the conversation-id parameter, and,
- the timeout value in the reply-by parameter must denote the latest time by which the sending agent would like to have received the next message in the protocol flow (not be confused with the latest time by which the interaction protocol should terminate).

## Conversation Identifier

Parameter	Description	Reserved Values
conversation-id	Introduces an expression (a conversation identifier) which is used to identify the ongoing sequence of communicative acts that together form a conversation.	

**Notes:** An agent may tag ACL messages with a conversation identifier to manage its communication strategies and activities. Typically this will allow an agent to identify individual conversations with multiple agents. It will also allow agents to reason across historical records of conversations.

It is required the usage of globally unique values for the conversation-id parameter in order to allow the participants to distinguish between several concurrent conversations. A simple mechanism to ensure uniqueness is the concatenation of the globally unique identifier of the sender agent to an identifier (for example, a progressive number) that is unique within the scope of the sender agent itself.

## Reply With

Parameter	Description	Reserved Values
reply-with	Introduces an expression that will be used by the responding agent to identify this message.	

**Notes:** The reply-with parameter is designed to be used to follow a conversation thread in a situation where multiple dialogues occur simultaneously. For example, if agent *i* sends to agent *j* a message which contains:

reply-with *<expr>*

Agent *j* will respond with a message containing:

in-reply-to <*expr*>

### In Reply To

Parameter	Description	Reserved Values
in-reply-to	Denotes an expression that references an earlier action to which this message is a reply.	

**Notes:** See notes for Section 2.5.3.

### Reply By

Parameter	Description	Reserved Values
reply-by	Denotes a time and/or date expression which indicates the latest time by which the sending agent would like to receive a reply.	

**Notes:** The time will be expressed according to the sender's view of the time on the sender's platform. The reply message can be identified in several ways: as the next sequential message in an interaction protocol, through the use of the reply-with parameter, through the use of a conversation-id and so forth. The way that the reply message is identified is determined by the agent implementer.