INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS

SISTEMAS MULTI AGENTES EMBEBIDOS APLICADO AL TRÁFICO URBANO

Trabajo final que para obtener el diploma de

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: Miguel A. Fernández García

Asesor: Raul Campos Rodríguez

Tlaquepaque, Jalisco, Octubre de 2016.

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS

SISTEMAS MULTI AGENTES EMBEBIDOS APLICADO AL TRÁFICO URBANO

Trabajo final que para obtener el diploma de

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: Miguel A. Fernández García
Becario CONACYT No. 424480

Asesor: Raul Campos Rodríguez

Tlaquepaque, Jalisco, Octubre de 2016.

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS

EMBEDDED MULTI-AGENT SYSTEMS APPLIED TO THE URBAN TRAFFIC

Final report to earn the diploma of

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presented by: Miguel A. Fernández García
CONACYT Scholarship No. 424480

Advisor: Raul Campos Rodríguez

Tlaquepaque, Jalisco. October, 2016.

Thanks to

Author sincerely wants to thank to:

- Thanks to my wife, for her patience and always being supportive on my goals which are also hers;
- Consejo Nacional de Ciencia y Tecnología (CONACYT) for granting me the opportunity of the scholarship tuition support. CONACYT Scholarship No. 424480;
- To ITESO for providing facilities, labs, and always being willing to help beyond academic topics;
- To my thesis advisor Dr. Raúl Campos Rodriguez, for supporting and providing professional coaching and helping with this milestone reached.

RESUMEN

Este trabajo presenta una implementación que aprovecha la Inteligencia Artificial Distribuida en la industria automotriz. *Agentes tipo Vehículo* obtienen información de *Agentes tipo Infraestructura* para decidir cuál es la mejor ruta basada en sus metas o prioridades.

El agente tipo vehículo proporciona un porcentaje a cada una de las metas que tiene y basado en el valor normalizado provisto por el agente tipo infraestructura, el vehículo calcula una *Utilidad* para cada ruta. El valor que resulte mayor corresponde a la mejor ruta para ese agente vehículo. Este sistema podría dar como resultado un tráfico balanceado y es uno de los puntos iniciales para tener una buena comunicación entre vehículos e infraestructura en **Ciudades Inteligentes**.

El **Internet de las cosas** es parte del diseño, debido a que es una tendencia tecnológica para autos conectados y ciudades inteligentes.

Los autos y la infraestructura como semáforos actúan como agentes. Éstos se comunican unos con otros usando “La Nube” a través del uso de **ACL** (Lenguaje de Comunicación de Agentes). Envían mensajes solicitando el estatus y basados en éste toman decisiones de cómo usar los recursos disponibles (por ejemplo, carreteras).

La infraestructura tiene información acerca de las rutas que está monitoreando. Cuando un vehículo tipo agente solicita información de una ruta específica, el agente tipo infraestructura informa el estatus de dicha ruta. Una vez que el vehículo tiene la información, evalúa cual ruta es mejor basado en sus metas o prioridades.

Un sistema Multi-Agente cooperativo puede ser usado para mejorar las rutas de manera dinámica y el manejo del tráfico.

El objetivo de esta implementación es tener un Sistema de Manejo de Tráfico eficiente en ciudades inteligentes.

La interacción de agentes en Sistemas Distribuidos ayuda a conseguir una meta común. En el caso del área automotriz, dicha interacción ayuda a encontrar una mejor ruta para un agente. Puesto que la información de diferentes rutas es compartida, el Sistema de Manejo de Tráfico trata de mantener un tráfico balanceado. Los agentes tipo vehículo tienen sus propias prioridades. Por ejemplo, es más importante para cierto agente la distancia que viajará y para otro el número de vueltas que hará en el mismo viaje.

La información del tráfico es actualmente usada para decidir si se usará una ruta o no. Sin embargo; la infraestructura no participa en el sistema para mantener el tráfico balanceado. La implementación descrita en este documento explica como un sistema distribuido cambia la perspectiva del tráfico en una ciudad y que tan importante es verla como parte de una ciudad inteligente donde todos los agentes juegan un papel importante.

El Sistema Multi-Agente está desarrollado en Java. Este lenguaje de programación facilita el uso del paradigma de Programación Orientada a Objetos. Por lo tanto; modularidad, escalabilidad y mantenibilidad son ventajas muy interesantes para este prototipo.

Los agentes son implementados con el uso de JADE (JAVA Agent Development Framework), un *framework* de software totalmente implementado en Java que cumple con las especificaciones de FIPA.

JADE provee una clase **Agente** que representa una base común para los agentes definidos posteriormente. Los agentes pueden ejecutar muchas tareas de manera concurrente en respuesta a diferentes eventos externos. Para lograr esto, los agentes JADE implementan objetos tipo *Behavior.*

Para corroborar que la implementación se puede hacer en una computadora y en un sistema embebido, se utiliza la tarjeta Intel Galileo Board Single. Desde esta tarjeta se corren agentes en un contenedor que se define en el host.

ABSTRACT

Distributed Artificial Intelligence (in Multi-Agent systems) is used to solve decision-making problems and techniques to solve city mobility issues. This work presents an implementation that takes advantage of the Distributed Artificial Intelligence in the automotive field. Vehicle Agents obtain information from Infrastructure Agents to decide what is the best route based on its goals or priorities. The vehicle agent gives weights to each of its goals and based on the normalized value provided by the infrastructure agent, it will calculate a Utility for each route. The resulted highest value corresponds to the best route for that vehicle agent. This approach could result on a balance in the traffic and one of the starting points for having a good communication between vehicles and infrastructure in Smart Cities

TABLE OF CONTENTS

1. INTRODUCTION 12

1.1. BACKGROUND 13
1.2. MULTI AGENT SYSTEMS 13
1.3. PROBLEM STATEMENT 14
1.4. OBJECTIVES 14
1.4.1. General Objective: 14
1.4.2. Specific Objectives: 14
1.5. PROPOSED APPROACH 15

2. STATE OF THE TECHNIQUE 16

2.1. MULTI AGENT SYSTEMS AND ARTIFICIAL DISTRIBUTED INTELLIGENCE 17

3. PROPOSED ARCHITECTURAL DESIGN 18

3.1. AUTOMOTIVE APPLICATIONS 19
3.2. MAS IN AUTOMOTIVE APPLICATIONS USE CASE DESCRIPTION 19
3.3. HARDWARE/SOFTWARE DESIGN 20
3.4. ROUTE ADAPTION FOR AGENTS ON A DISTRIBUTED SYSTEM 20
3.5. DISTRIBUTED ARCHITECTURAL DESIGN 21
3.6. ARCHITECTURAL DESIGN LAYOUT 22
3.7. MAS SOFTWARE FRAMEWORK 23
3.7.1. AGENT BEHAVIORS 25

4. IMPLEMENTATION 28

4.1. IMPLEMENTATION OF AN ONTOLOGY 29
4.2. IMPLEMENTATION OF AN ONTOLOGY 30
4.3. SETTING UP THE WORKBENCH ENVIRONMENT 30
4.3.1. HOST MACHINE 31
4.4. EMBEDDED BOARDS FOR MAS AGENT SOLUTION 32
4.4.1. PASSING ARGUMENTS TO AGENTS 33
4.4.2. RUNNING SEVERAL AGENTS WITH A BASH SCRIPT 34

5. TEST AND RESULTS 35

5.1. DESCRIPTION OF THE EXPERIMENTATION SCENARIO 36
5.1. RESULTS 37

6. CONCLUSIONS 40

6.1. CONCLUSIONS AND DISCUSSIONS 41

FIGURES LIST

Figure 1. Leader Election Conceptual Layout 17
Figure 2. Leader Election Conceptual Layout. (The Bully Election Algorithm) 26
Figure 3. **Swarm Optimization Surface (Particle Swarm Optimization Algorithm)** 27
Figure 4. **Architectural Design Layout** 22
Figure 5. **Conceptual Diagram of the Agent's Interaction** 23
Figure 6: Agent Management Reference Model 24
Figure 7: Agent Life Cycle accordingly to JADE 24
Figure 8: UML Model of the Behavior class hierarchy 25
Figure 9: Class Diagram of the Proposed Solution 29
Figure 10: Class Diagram of the Proposed Solution 32
Figure 11: Development Boards. Left: Edison Board. Right: Galileo Board 33
Figure 12. Running Agents 59
Figure 13. Vehicle Weights 59
Figure 14. Running Vehicle 2 60
Figure 15. Example of agents' communication 60

TABLES LIST

Table 1. Agents Parameters for the Scenario 36
Table 2. Vehicle Weights 37
Table 3. Route_1 Utilities 37
Table 4. Route_2 Utilities 38
Table 5. Route_3 Utilities 38
Table 6. Route_4 Utilities 38

ACRONYMS AND SYMBOLS

IoT	Internet of Things
GPS	Global Positioning System
TMS	Traffic Management System
JADE	Java Agent Development Framework
FIPA	Foundation of Intelligent Physical Agents
SSH	Secure Shell
PSO	Particle Swarm Optimization
GUI	Graphical User Interface
JDK	Java Development Kit
ACL	Agent Communication Language
MAS	Multi Agent System
DAI	Distributed Artificial Intelligence
LPP	Linear Programming Problem

1. INTRODUCTION

Abstract: *This chapter briefly presents the background of the object of study, problem definition and justification.*

1.1. Background

This work deals with a proposal of project that proofs a concept for Automotive Applications. Distributed Artificial Intelligence (in Multi-Agent systems) is used to solve decision-making problems and techniques to solve city mobility issues.

Behavior of animals has been studied for hundreds of years. One reason to do it is to replicate the comportment of them, on different technologic devices and systems.

Since nature has been always intelligent enough to make an ecosystem to work in an organized way, humans learn from it and apply concepts in the technology field. One example is smart cities, where people need new technologies to have a better quality of life in terms of mobility.

This work pretends to show an application of a Multi-Agent System to resolve problems of traffic considering the priorities that each vehicle has when moving from point A to point B.

Distributed Artificial Intelligence is used to solve decision-making problems and city mobility issues. **Leader Election** process and **Particle Swarm Optimization** technique are examples of available resources that can be used on intelligent distributed systems.

1.2. Multi Agent Systems

There are several techniques and algorithms that can be used to solve problems in Distributed Artificial Intelligence (DAI), two of them are the following:

- Leader Election
- Particle Swarm Optimization (PSO)

There are reasons for studying distributed artificial intelligence applied on automotive field. For example, autonomous driving technology is part of the automotive industry strategy.[1] One can find information on the internet about the interest of several brands on this trend. In addition, not only companies involved in the automotive manufacturing process want to take part of this upcoming technology. Industries like social networking, information technology and infotainment among others are investing on the research and development of it.

People also want to have a better quality of life by reducing the time they spend in the car to travel from one place to another.

[1] (Continental, s.f.)

The work presented on this document considers the reasons mentioned above.

1.3. Problem Statement

Persons have alternatives when they want to go from a hypothetical point A to point B. The options include driving by car, taking public transportation, walking and going by bicycle, to mention few examples. There are environment conditions that take part of the experience that the person will face.

For example, if the person decides to walk, they will expect not to have rain and to have a good condition of the streets to be walked.

If a car will be used for the trip, the individual might use a **GPS** or a web mapping service to find "the best route". However; there are specific conditions that the mentioned systems do not consider when calculating routes. In addition, preferences might differ from one user to another. They could even change when already on the trip and the "best route" is likely to be different.

As it was mentioned already, the system is not limited to the current available technologies. When autonomous driving becomes a reality, the number of shared vehicles will increase and the users of them will not be necessarily the owners.

Thus, strategies for coordination, negotiation, interaction and others among autonomous vehicles will be required. This work proposes the Multi Agent System approach to cope with the challenge in the future scenarios of automotive driving.

1.4. Objectives

1.4.1. General Objective:

- The general objective of this work is to apply a Multi Agent Systems approach to the solution for the load balancing problem in automotive traffic useful for intelligent traffic control in smart cities.

1.4.2. Specific Objectives:

- To use the concept of Agent for the modeling of cars and infrastructure (routes)
- To define the required concepts within an suitable ontology for the communication of the Agents
- To implement a distributed load balancing policy among the car Agents and infrastructure Agents

- To experiment with methods that considers autonomous vehicles preferences and route's conditions to calculate the best route and infrastructure objectives.

1.5. Proposed Approach

This work considers a method to calculate the best route for a vehicle based on its priorities. It considers the autonomous driving technology and its necessity of sharing information continuously.

The interaction of agents in a Distributed System helps to achieve a common goal. In the case of the automotive field, such interaction helps to find a better route for an agent. As far as information of different routes is shared, the **Traffic Management System** tries to maintain a balanced traffic. The Vehicle Agents have their own priorities. For example, it is more important for a certain type of vehicles the distance it will travel and for another one the number of turns it will make on its travel. Consider for example a big cargo truck versus a utilitarian car.

Information about traffic is currently used to decide whether to use a certain route or not. However; infrastructure typically does not take part of a system to keep the traffic balanced. This work considers that the infrastructure could play an important role in the load balancing strategy. For example, the infrastructure may consider information about building constructions in certain areas. Thus, an objective of the infrastructure could be to reduce the traffic flow in such areas.

The implementation described on this document explains how a distributed system changes the perspective of the traffic in a city, and how important is to see it as part of a smart city where all agents play an important role.

2. STATE OF THE TECHNIQUE

Abstract: *This chapter briefly presents the related techniques and technologies which are relevant for the context of this work.*

2.1. Multi Agent Systems and Artificial Distributed Intelligence

The Multi Agent Systems (MAS) is a paradigm in the Computer Sciences and related areas where a system of interest is conceived as a set of autonomous entities called agents, as well as its interaction mechanisms. The agent is considered to be an autonomous entity with the ability to "sense" the environment thorough a set of physical or logical sensors and to "interact" or modify such an environment by a set of physical or logical actuators, as well. A kind of "intelligence" or "inference" mechanism is also conferred to an agent. Thus, its actions to the environment are based on the sensors and the inference machinery.

The interaction among agents is generally considered as message passing based on a well-structured interaction protocols. The content of the message is "information" that may lie in a particular context called ontology. The Figure 1 depicts a general layout of a MAS.

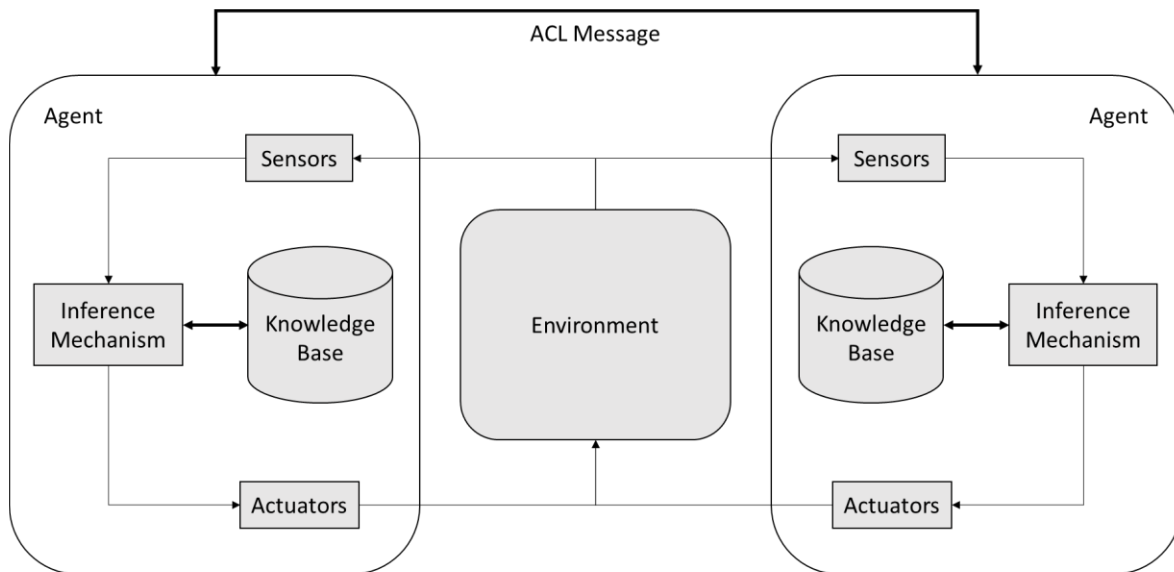

Figure 1. Leader Election Conceptual Layout

The field of the MAS has proved to be a suitable solution for problems of distributed nature, where the information, the control, the processing or all of them are not centralized but rather distributed. Thus, a set of problem has been well studied and useful solutions obtained.

Some problems from the MAS field, which were useful in this work, are briefly reviewed next.

3. PROPOSED ARCHITECTURAL DESIGN

Abstract: *This chapter shows the architecture of the proposed solution and briefly discusses the technologies involved in the implementations.*

3.1. Automotive Applications

The automotive industry is moving towards the automated mobility. In order to achieve the goal of making mobility safer and having an optimized system for moving people in the world, a visionary technology is needed.

Some applications of this work is route decision making to avoid traffic jams, car and infrastructure coordination to load balancing, among others. In order to proof the concept, the techniques mentioned above are needed; automobiles have the need to find out what comes ahead, how many of them are heading the same direction and the mobility agent-based system should decide what route is better for all of the agents, and not only for one of them.

3.2. MAS in Automotive Applications Use Case Description

The aim of the project described in this document is to cover the following scenario:

1. The prototype shall simulate agents as part of a distributed system.
2. An embedded platform could be used for the implemented agents, for example, the **Intel Galileo or Edison Development Boards**.
3. The system considers two types of agents: **infrastructure agents** (e.g. Semaphores, routes, highways, etc.) and **vehicle agents** (e.g. Cars, cargo trucks, ambulances, etc.).
4. It is considered that the infrastructure agents shall provide information about the conditions of different routes and the vehicle agents shall decide which route is more convenient based on its priorities or goals.
5. Vehicle agents shall base its decision on driver’s goals. For example, but not limited, to the following goals:
    a. Minimize Travel Time
    b. Minimize Travel Distance
    c. Minimize/Maximize Arterial Streets
    d. Minimize Number of turns
    e. Minimize/Maximize Roadway classification changes
6. Infrastructure agents shall provide a linear normalized weighted for each of the goals that the vehicle agents have.

7. Based on the received data, vehicle agents shall calculate the utility as follows[2]:

$$U^{ip} = \sum_{g} W_g^i N_g^{ip}$$

Such that:

$$\sum_{g} W_g^i = 100$$

Where i stands for the $i - th$ agent, p is the specific path, g is the specific goal (see previous point 5), U is the overall utility function, W is the weight conferred to specific goal by $i - th$ agent and N is the normalized score for goal g by $i - th$ agent.

8. The route with the maximum utility shall be selected by the vehicle agent as the best route.

3.3. Hardware/Software Design

The presented design is based on Distributed Systems applied to the Automotive Industry. **Internet of Things** (IoT)[3] is part of the design, as it is a trending technology for **connected cars** and **smart cities**.

The cars and infrastructure devices like Semaphores act as agents. They are communicating with each other by using the **Cloud Network** through the usage of **ACL**. They send messages to know the status and based on that make decisions on how to use the available resources (i.e. roads).

Infrastructure devices have information about routes they are monitoring. When a vehicle agent requests information about a specific route, the infrastructure device informs the status of such route. Once the vehicle has the information, it evaluates which route is the best based on its goals.

A cooperative, distributed multi-agent system can be used to improve dynamic routing and traffic management.[4]

3.4. Route Adaption for Agents on a Distributed System

The interaction of agents in a Distributed System helps to achieve a common goal. In the case of the automotive field, such interaction helps to adapt the route that the vehicle is following by sharing

[2] (Adler, et al., n.d.)
[3] (Mann, 2015)
[4] (Adler, et al., n.d.)

information like **speed** and **location**. Therefore, the spacing between vehicles is known and the traffic density can be determined:

- $k = \frac{1}{s}$
- $k = number\ of\ vehicles\ per\ unit\ length\ of\ the\ road$
- $s = center\ to\ center\ distance\ between\ two\ vehicles$

Agents (vehicles, in this case) calculate a several routes to go from point A to point B. Each agent takes the fastest route at that moment. However; such route might not be the fastest afterwards.

If **jam density** is detected, the agent will set a flag of traffic jam in the corresponding area. If other agents are heading that area, they should take different routes in a balanced way in order to avoid creating another traffic jam:

The number of agents heading the same direction is divided by the quantity of available alternative routes. The result is the quantity of cars that will follow each alternative route.

3.5. Distributed Architectural Design

The proposed architecture considers the following aspects:

- The number of agents in the distributed system is arbitrary. That is, it could be 2 agents, i.e. 1 car and 1 route, to more than 30 agents.
- The agents of the system shall be implemented on an embedded board, e.g. the **Intel Galileo Board Single** hardware.
- The architecture distinguishes two types of agents: **unsteady** (e.g. routes) and **steady** (e.g. Cars).
- The architecture considers a load balancing algorithm among the car Agents and route Agents.
- The architecture considers that the route Agents shall send their parameters of interest to all the car Agents that request them.
- The car Agents use the information provided by the route Agents to calculate its best route.
- The distributed load balancing algorithm considers the infrastructure requirements, for example, to keep some route under some peek value of traffic density.

3.6. Architectural Design Layout

The Figure 4 depicts a layout of the conceptual architectural design. The agents, i.e. cars, semaphores, highways are interconnected by an infrastructure network. The design of such interconnecting network is out of the scope of this work. Thus, it is an assumption that a communication mechanisms exists in the system.

Semaphores, for example, are on its own agents that have a set of goals. They continuously send the status of the road they are monitoring. If the density of such road is very high, for instance, then the unsteady agents decide whether to proceed that way or not, to mention one scenario.

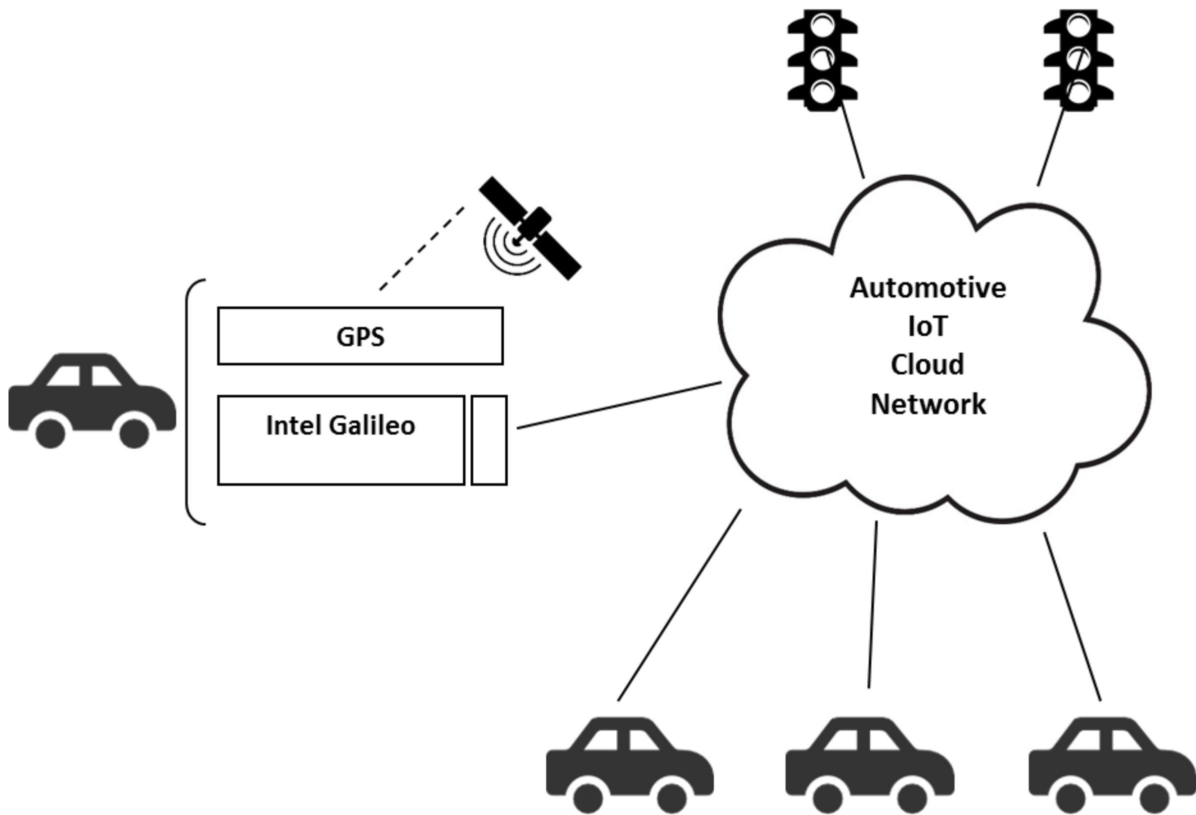

Figure 2. **Architectural Design Layout**

The Figure 5 provides a conceptual diagram of the agent's interaction in the proposed architecture. Vehicle agents "request" information about the "status" of the infrastructure by asking to the proper agent. With the information of the nearby ways, the car agents are able to decide which one provides the best solution for the driver's goals.

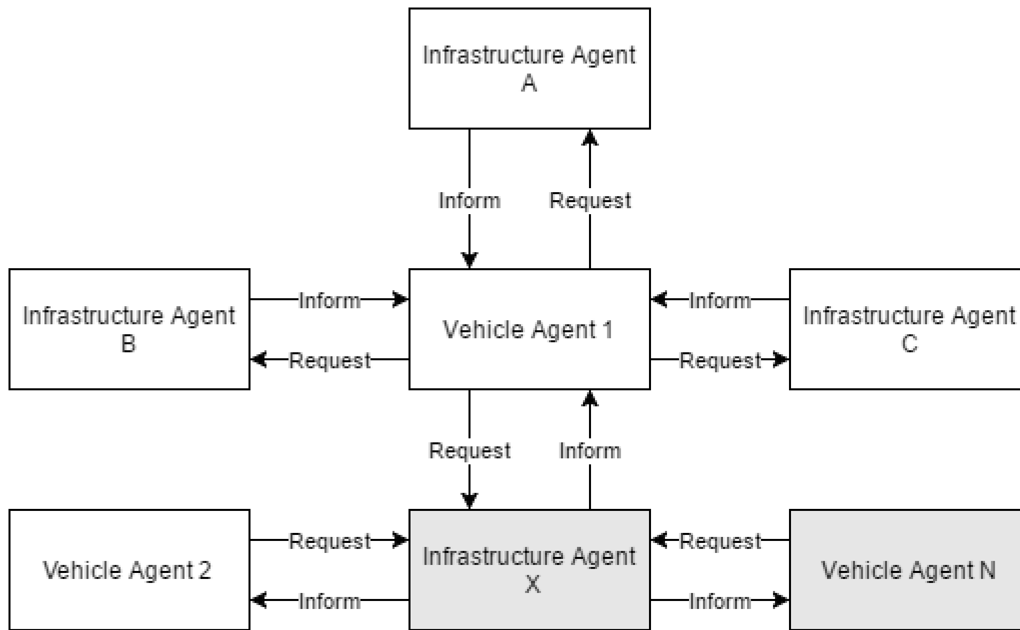

Figure 3. **Conceptual Diagram of the Agent's Interaction**

3.7. MAS Software Framework

The Multi-Agent based system is made on **Java**. This programming language facilitates the use of **Object Oriented Programming** paradigm. Hence; *modularity*, *scalability* and *maintainability* are very interesting advantages for this prototype.

Agents are implemented with the use of **JADE** (JAVA Agent Development Framework), a software framework fully implemented in Java that complies with the **FIPA** Specifications.

Agent Management Reference Model according to FIPA[5]. The Figure 6 depicts a layout of a MAS based on the JADE infrastructure.

[5] (FIPA, 2004)

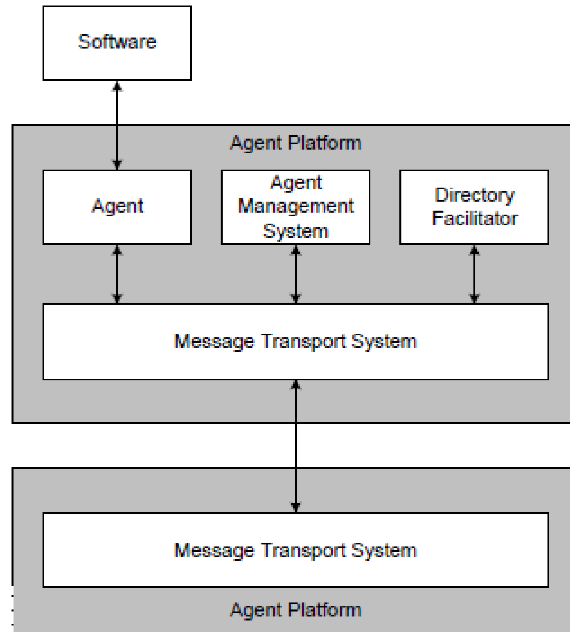

Figure 4: Agent Management Reference Model

JADE provides an Agent class that represents a common base for the further defined agents. The agents have a life cycle also specified by FIPA. The Figure 7 depicts the transition state machine of agent life cycle.

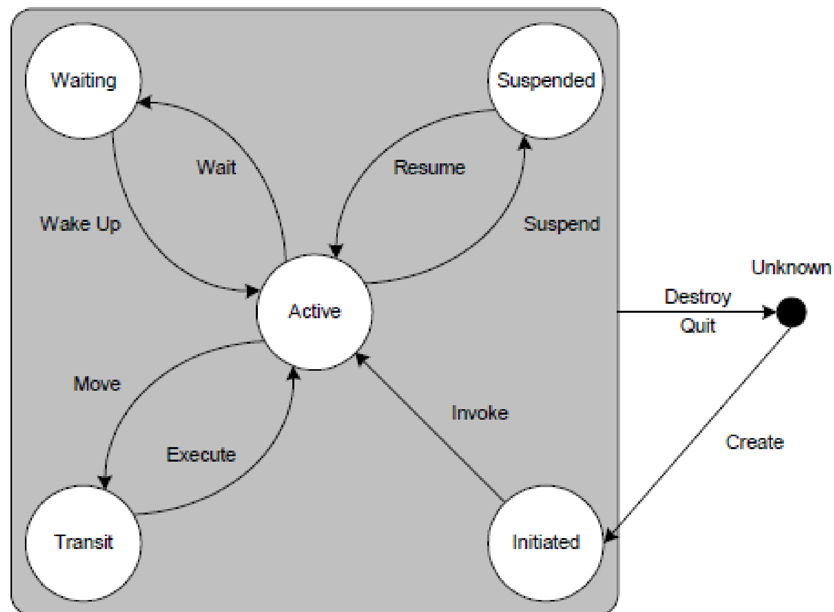

Figure 5: Agent Life Cycle accordingly to JADE

3.7.1. Agent Behaviors

Agents can execute several concurrent tasks in response to different external events. To accomplish this, JADE agents implement Behavior objects.

This Figure 8 shows the UML class diagram for JADE behaviors, as specified in the JADE programmers guide[6].

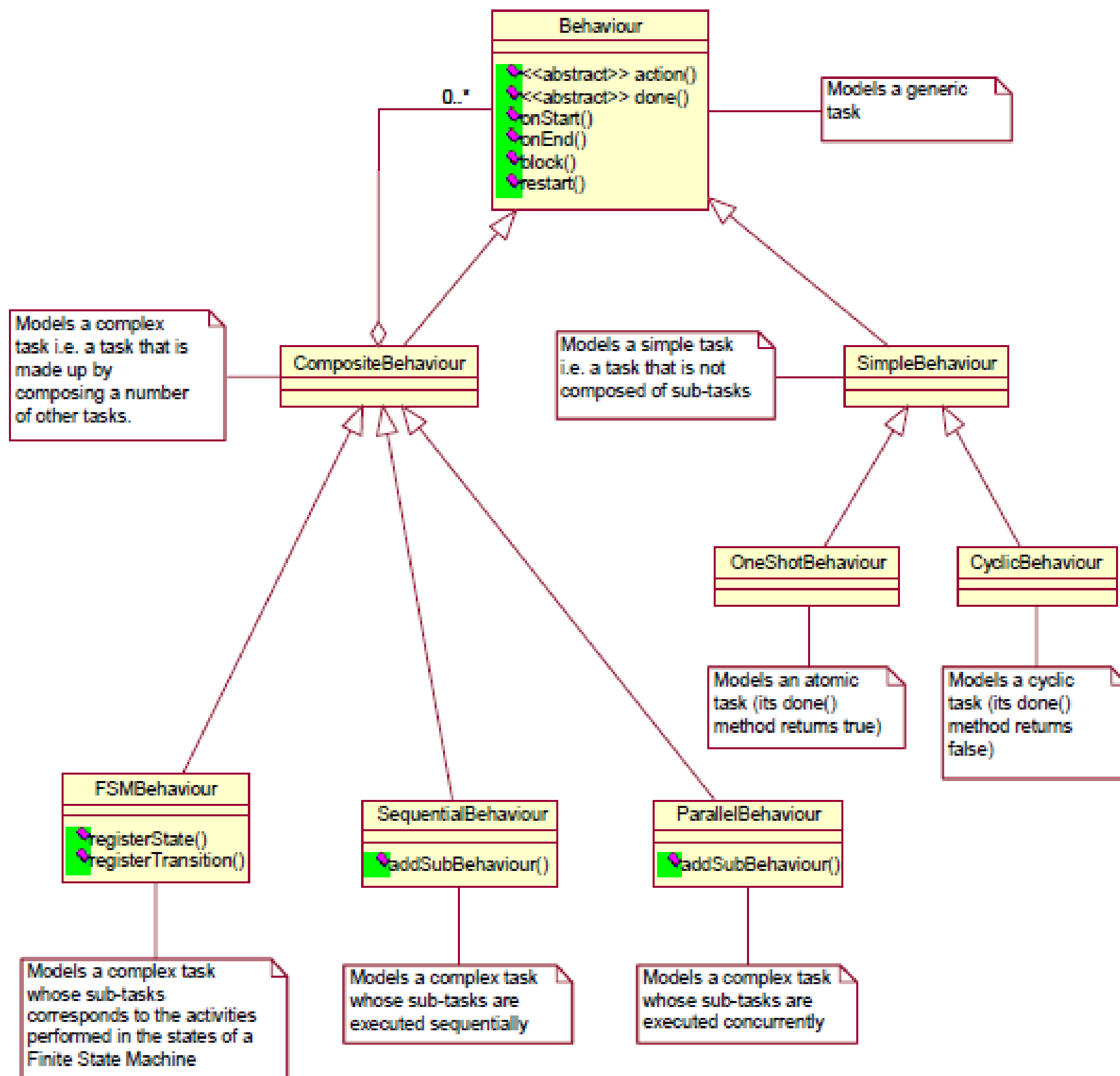

Figure 6: UML Model of the Behavior class hierarchy

[6] (JADE, 2010)

3.8. Leader Election Problem

In distributed systems and depending on the implementation technology, nodes or agents communicate with each other using shared memory or via messages. In order to perform a system's task, such agents usually require some kind of coordination.

However; one of the principles of the MAS is that the agents should not rely on one of them to organize the entire system. They instead should implement a process for electing a Leader, for example. In this way, if an elected leader abandons the system, a new one could be elected.

Based on this asseveration, all nodes should implement the same software and be able to elect, and to be a leader in multi-agent systems despite of the change in quantity and separation or joining of nodes. Consider the conceptual layout of the Figure 2.

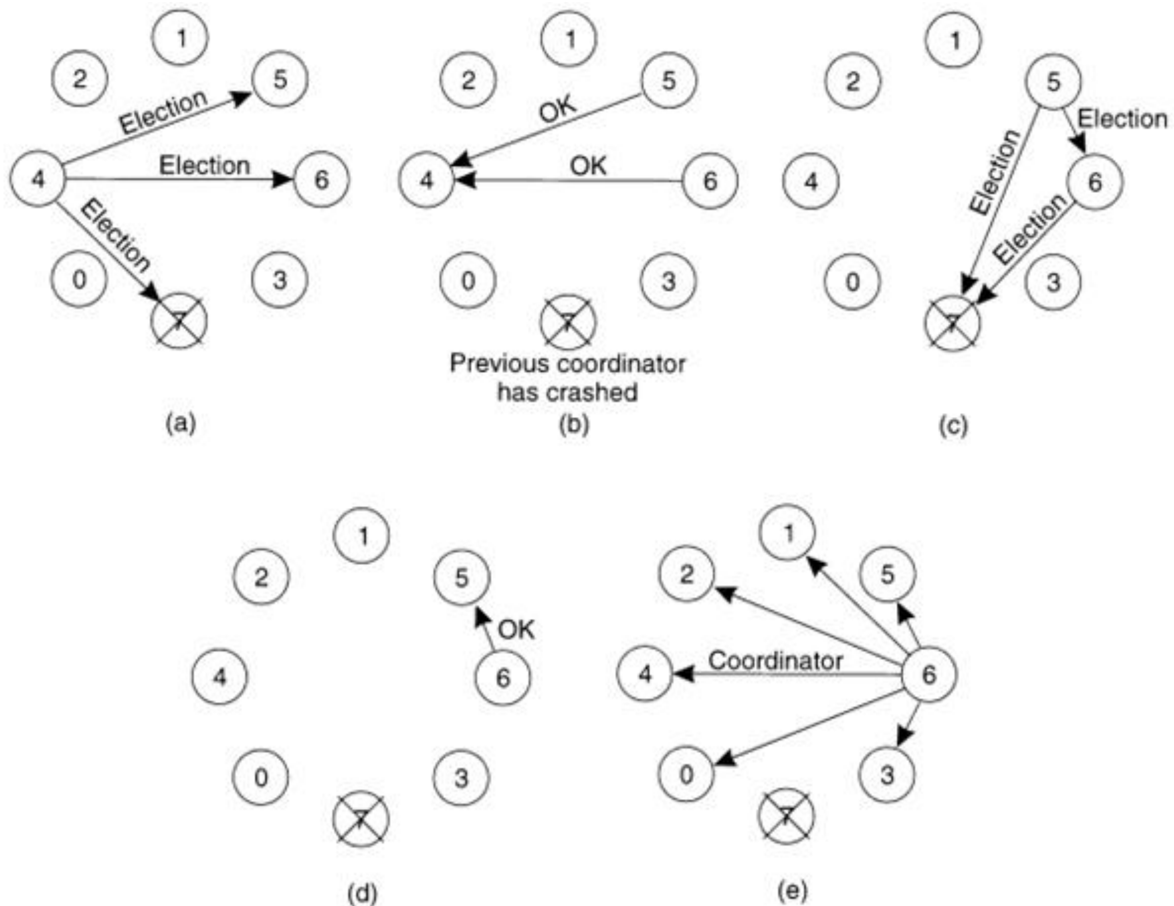

Figure 7. Leader Election Conceptual Layout. (The Bully Election Algorithm[7])

[7] (Kaminsky, 2013)

3.1. Particle Swarm Optimization

This technique is inspired in social behavior of bird flocking or fish schooling.

The method optimizes a problem solution by iteratively improving a candidate solution. The candidate solutions (particles) are moved around in the search space, this process will provide the best solution according to the maximum iterations or minimum error criteria.

Figure 2 shows a graph that illustrates how the particles are located in the system. Taking into account that the solution is in the center, the particles will be moved towards it in order to find the solution.

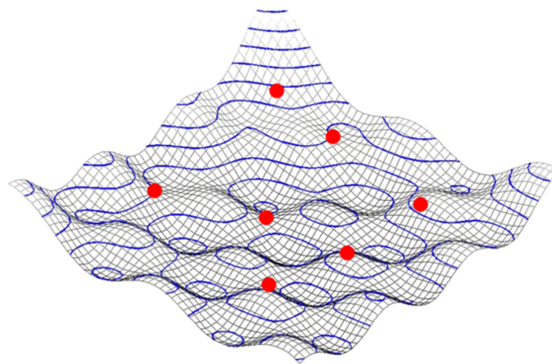

Figure 8. Swarm Optimization Surface (Particle Swarm Optimization Algorithm[8])

3.2. Web mapping and GPS devices

There are features on web mapping services like Google to know the traffic conditions in real time. The method used is GPS-determined location of mobile phones sends anonymous information and based on that it is determined how fast they move. This could result on inaccurate data because the infrastructure is not participating on informing its conditions and the method is strongly dependent on the quantity of mobile phone users with "My Location" enabled on their devices.9

One of the main aims of this work is to provide a method that finds the best route for a vehicle, based on the user's preferences and at the same time meet the requirements of the infrastructure, represented by routes, semaphores, highways, etc.

[8] (Institue of Engineering and Computational Mechanics, s.f.)

[9] (Dave Barth, 2009)

4. IMPLEMENTATION

Abstract: *This chapter describes the main aspect of the agents, behaviors, message passing and ontology implemented in this work.*

4.1. Implementation of an Ontology

The Figure 9 depicts a class diagram of the proposed solution. It shows the Agent class and its behaviors. Additionally, it illustrates the predicate, ontology and message implemented.

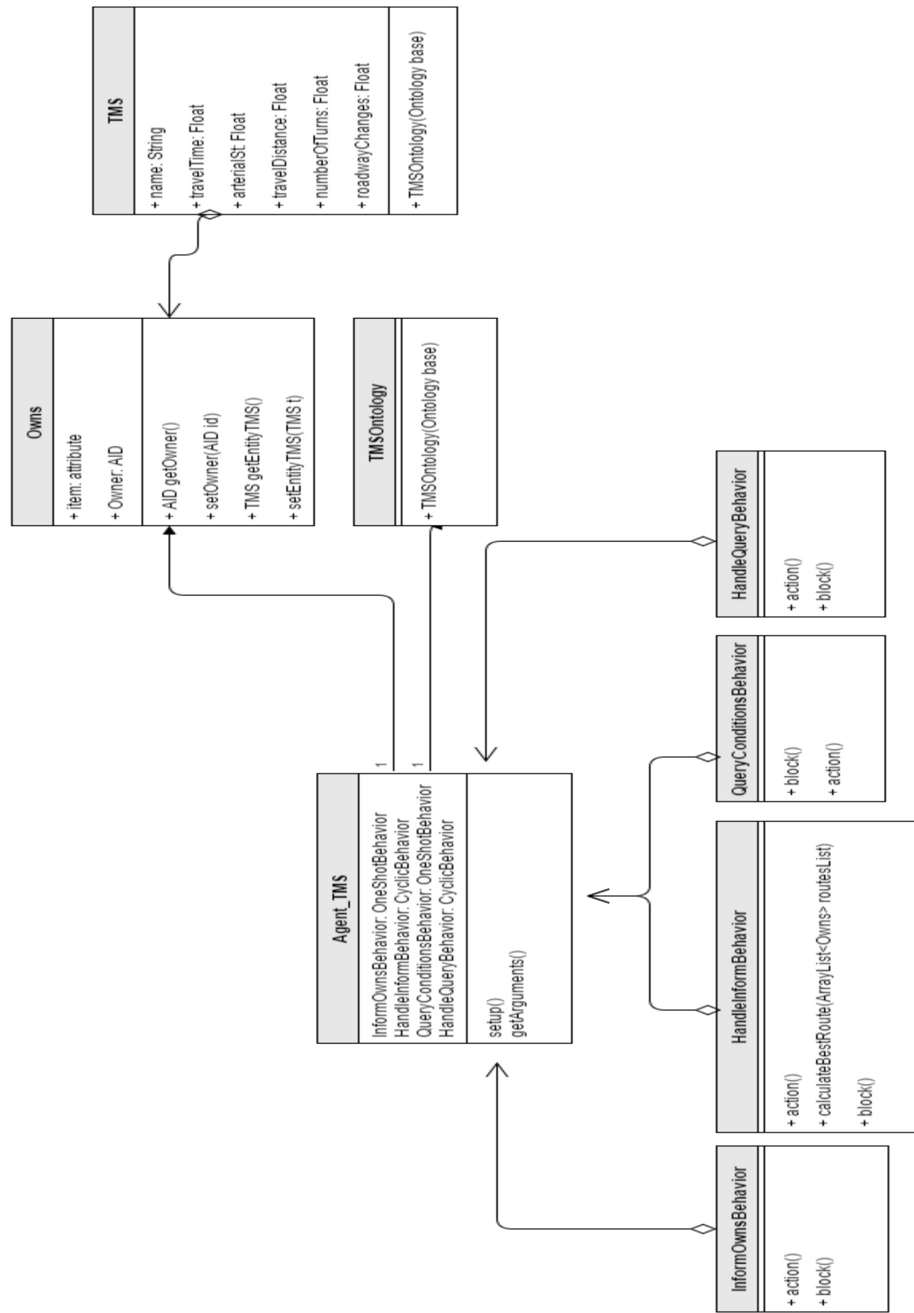

Figure 9: Class Diagram of the Proposed Solution

The main classes are described as follows:

- **TMS Class**: The Traffic Management System Class implements the information that the Infrastructure Agent provides to the Vehicle Agent.
- **Owns**: This class is a **predicate**. It is used for instance as the content of an INFORM message. It contains the TMS Class and it is used to send the data requested by the Vehicle Agent. TMSOntology was implemented specifically for this purpose.

4.2. Implementation of an Ontology

An **Ontology** includes those elements that agents are using within a specific content of the messages in the context of a specific application. Generally speaking, it defines a vocabulary and the relationships between them in the context of the application.[10]

In the solution proposed and implemented in this work, the TMSOntology adds the generic Owns **predicate**, TMS **concept** and the next **primitive** schemas:

- TMS_NAME: Name of the agent that holds the information.
- TMS_TRAVELTIME: Actual time from origin to destination.
- TMS_TRAVELDISTANCE: Distance from origin to destination.
- TMS_ARTERIAL_ST: Distance on surface streets/total distance traveled.
- TMS_NUMBEROFTURNS: Road Changes.
- TMS_ROADWAYCHANGES: Going from highway to arterial or three-lane to two-lane, etc.

4.3. Setting up the Workbench Environment

The system was tested using **Ubuntu 16.04** as the host machine where the GUI of the JADE framework is displayed for debugging and testing purposes. An **Intel Galileo Board Single and Edison Development Board** has been used as the hardware support for the agents in the system.

[10] (JADE, 2010)

4.3.1. Host Machine

First, JADE should be downloaded from the corresponding website: http://jade.tilab.com/download/jade/[11]. The **jadeAll** package should be placed somewhere in the Linux filesystem. It is also needed the **Miscellaneous** Add-On available at http://jade.tilab.com/download/add-ons/. This last package contains the implementation of the Leader Election algorithm.

To send the JADE commands, the java source files must be compiled. It is needed to have **JDK** (Java Development Kit) installed in the host machine.

If the *classpath* environment variable has the correct value, it is possible to compile a java source file using the shell and executing the next line:

```
$ javac Agent_TMS.java
```

An Agent_TMS.class and other dependent class files should be generated on the same folder. To open the JADE GUI, the next command should be sent:

```
java jade.Boot -gui -services \
"jade.core.messaging.TopicManagementService;\
jade.core.event.NotificationService;jade.core.mobility.AgentMobilityService" -agents
"Vehicle_1:Agent_TMS(55, 10, 5, 5, 25, true, 0)"
```

This line indicates that the **GUI** should be opened, as shown in the Figure 10. The specified services for the broadcast message passing and notification facilities should be started, as well. The *-agents* argument is used to indicate the **Name**, the **Class** to be called and its arguments (explained in section 3.3.1).

[11] (JADE, 2010)

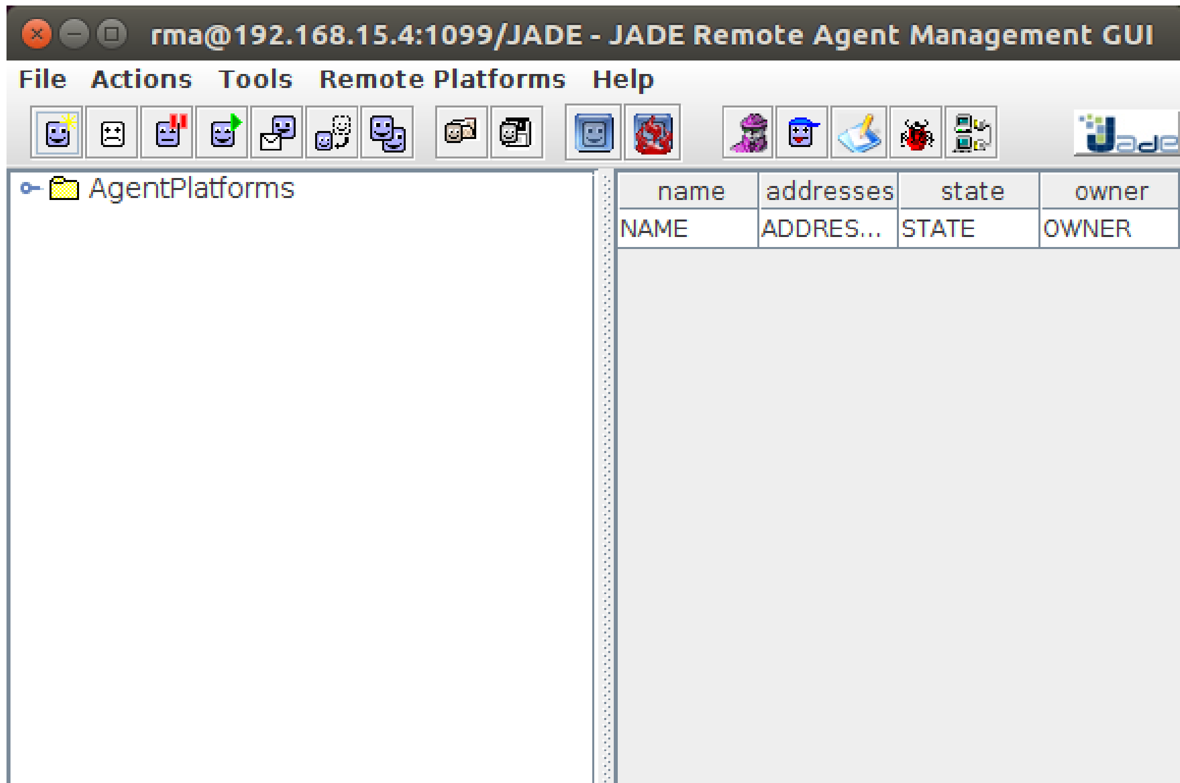

Figure 10: Class Diagram of the Proposed Solution

4.4. Embedded Boards for MAS Agent Solution

Intel provides a set of high performance development board such as the Galileo and Edison Boards, which target market are the IoT, makers, students and professionals. These boards are on its own a computer of small size that could be powered by battery and that are able to run Linux distros[12].

These boards are a good option for implementing one or several agents. It requires a micro-SD card that should be previously made bootable as indicated in the getting started Galileo/Edison. The tutorial are quite illustrative and a ready to use material. Some of the steps for the setup of the boards have been omitted here, but an interested reader may refer to the following links for more information:

- https://software.intel.com/en-us/articles/getting-started-galileo-arduino.
- https://software.intel.com/en-us/get-started-edison-windows

The board can be programmed using C++ or Java with the Intel System Studio IoT Edition. Python can also be used directly on the board.

[12] (Intel, 2014)

For the implementation described on this work, the bootable micro-SD with Linux (i.e. Ubuntu 14.04) installed will be enough.

SSH can be used to connect to the board from the host, by opening a session with the board being both on the same network:

$ ssh root@192.168.15.7

The Figure 11 shows the Intel Edison and Galileo Development boards. These boards run a Linux distribution provided by Intel. A JDK is configured and running over the Linux in the boards. The JADE framework is then configured to run over the JDK. This JADE framework is the environment where the developed agents live.

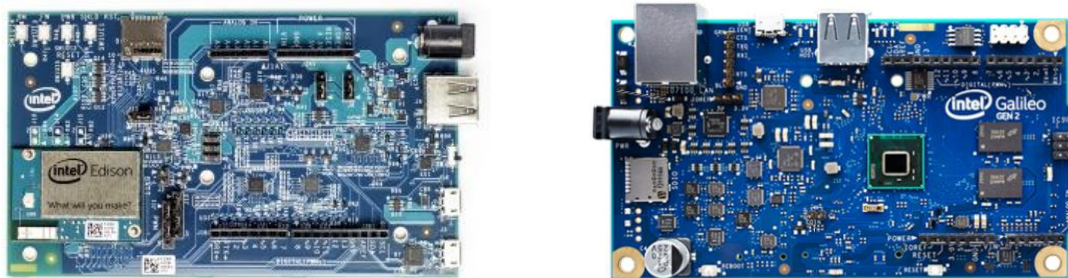

Figure 11: Development Boards. Left: Edison Board. Right: Galileo Board

4.4.1. Passing Arguments to Agents

The Agent can receive arguments and they can be obtained by calling the method *getArguments()*[13]. For the current implementation, the arguments that the agent receives are the following:

- Travel Time weight in percentage
- Travel Distance weight in percentage
- Arterial Streets weight in percentage
- Turns weight in percentage
- Roadway Changes weight in percentage

[13] (Fabio Bellifemine, 2010)

- Vehicle? True or False (to indicate the type of agent)
- Route Number: this value is used to add a route to the agent if it is of infrastructure type

As far as the same source code is used for a Vehicle Agent and an Infrastructure Agent, some of the parameters will be ignored if they do are not valid for the indicated agent.

4.4.2. Running several agents with a bash script

Several agents can be run using a bash script (found in the **Appendix 1**). Only one of the agents should run the JADE GUI (using *-gui*), otherwise an error will be raised.

5. TEST AND RESULTS

Abstract: *This chapter describes the results and testing of the implementations developed in this work.*

5.1. Description of the Experimentation Scenario

This experiment considers that there are four routes available, each one known by an infrastructure agent. There are two vehicles that want to receive information from such routes. According to Distributed Systems, they will “born” with some attributes that will receive through the arguments described on the next table:

Table 1. Agents Parameters for the Scenario

	Arguments						
Agent Name	Travel Time	Travel Distance	Arterial Streets	Turns	Roadway Changes	Vehicle?	Route Number
Route_1	0	0	0	0	0	FALSE	1
Route_2	0	0	0	0	0	FALSE	2
Route_3	0	0	0	0	0	FALSE	3
Route_4	0	0	0	0	0	FALSE	4
Vehicle_1	55	10	5	5	25	TRUE	0
Vehicle_2	5	5	80	5	5	TRUE	0

The Table 1 indicates that six agents in total will be run. The first five arguments represent the weight (importance) that each agent gives to that goal. The first four agents representing routes have a zero value on those arguments because they do not have such goals. Their job is to inform those conditions to the vehicle agents.

The sixth argument, i.e. *Vehicle?* is used to indicate whether the agent is a vehicle or an infrastructure agent.

Finally, the seventh argument indicates the route number. This value only concerns the infrastructure agents and its objective is to have a unique ID for each route.

In the current implementation, the vehicle agents ask for the Normalized values of each condition every 10 seconds. As soon as it receives the values of each route, it calculates the best route based on the weights given when it was born.

The vehicle has a list of routes, in case there is a new one, it will add such route to the Array List *routesList*, defined as a global variable:

- private ArrayList<Owns> routesList = new ArrayList<Owns>();

For testing purposes, the normalized values of the routes where simulated as follows:

```
public static Float[][] SIM_VALUES = new Float [][]
{{0.11f, 0.52f, 0.69f, 0.88f, 0.45f},
              {0.88f, 0.12f, 0.12f, 0.73f, 0.99f},
              {0.23f, 0.22f, 0.88f, 0.43f, 0.25f},
              {0.44f, 0.43f, 0.19f, 0.25f, 0.81f}};
```

The values above where used to make the calculations manually and verify whether the implementation is providing the expected values. This verification can be found in the *Results* section.

5.1. Results

The calculations were made in analytically in order to compare against the results computed by the car agents. The Table 2 shows the weights that each agents assigns to each goal.

Table 2. Vehicle Weights

Priorities/Goals	Vehicle_1 Weights	Vehicle_2 Weights
Travel Time	55%	5%
Travel Distance	10%	5%
Arterial Streets	5%	80%
Number of Turns	5%	5%

The following tables show the selection of the results of the utility of the routes in the experiments. These results agree with the expected values accordingly with the weights of the agents.

Table 3. Route_1 Utilities

Route_1	V1 Utility	V2 Utility
0.11	0.0605	0.0055
0.52	0.052	0.026
0.69	0.0345	0.552
0.88	0.044	0.044
0.45	0.1125	0.0225
	0.3035	0.65

Table 4. Route_2 Utilities

Route_2	V1 Utility	V2 Utility
0.88	0.484	0.044
0.12	0.012	0.006
0.12	0.006	0.096
0.73	0.0365	0.0365
0.99	0.2475	0.0495
	0.786	0.232

Table 5. Route_3 Utilities

Route_3	V1 Utility	V2 Utility
0.23	0.1265	0.0115
0.22	0.022	0.011
0.88	0.044	0.704
0.43	0.0215	0.0215
0.25	0.0625	0.0125
	0.2765	0.7605

Table 6. Route_4 Utilities

Route_4	V1 Utility	V2 Utility
0.44	0.242	0.022
0.43	0.043	0.0215
0.19	0.0095	0.152
0.25	0.0125	0.0125
0.81	0.2025	0.0405
	0.5095	0.2485

The agents obtain the maximum of all the routes, the result will be the best route. In this case, **Route_2** will be the best for **Vehicle_1** with a total utility of **0.786** and **Route_3** will be the best for **Vehicle_2** with a total utility of **0.760**.

Screenshots of the execution of the tests can be found on **Appendix 2**.

6. CONCLUSIONS

6.1. Conclusions and Discussions

This work put together a set of different types of knowledge and experiences. In one side, Linux and its importance were noticed. On the other one, the considerations of an embedded system will be always present.

Distributed Artificial Intelligence is an area well explored in computer science field. However; it is not very known in other industries like the automotive or aeronautics. The results of this work show that the journey in autonomous driving has just begun and many areas of study will be useful in the very near future.

A lot of information can be shared between agents on a distributed system. This could help considerably when trying to organize a city with the use of technology. Smart cities are a good example, we might have Internet of Things available in semaphores and traffic signs, but it will not be useful if we do not know how to share information.

The automotive industry and universities should participate on the development of efficient systems to have a better quality of life using technology.

REFERENCES

Adler, J. L. et al., n.d. *A multi-agent approach to cooperative traffic management and route guidance.* [Online]
Available at: www.sciencedirect.com

Continental, n.d. *2025 AD.* [Online]
Available at: https://www.2025ad.com/
[Accessed 2016].

Dave Barth, P. M. f. G. M., 2009. *Google Blog.* [Online]
Available at: https://googleblog.blogspot.mx/2009/08/bright-side-of-sitting-in-traffic.html
[Accessed 2016].

Fabio Bellifemine, G. C. T. T. G. R., 2010. *Jade Programmer's Guide,* s.l.: Tilab.

FIPA, 2004. *FIPA Agent Management Specification.* [Online]
Available at: www.fipa.org

Institue of Engineering and Computational Mechanics, n.d. *ITM University of Stuttgart.* [Online]
Available at: http://www.itm.uni-stuttgart.de/research/pso_opt/pso_anim_en.php
[Accessed 2016].

Intel, 2014. *Intel Galileo Board User Guide,* s.l.: s.n.

Intel, n.d. *Getting Started Galileo.* [Online]
Available at: https://software.intel.com/en-us/articles/getting-started-galileo-arduino
[Accessed 2016].

JADE, 2010. *JAVA Agent Development Framework.* [Online]
Available at: http://jade.tilab.com/

Kaminsky, A., 2013. *Lecture Notes -- Module 4. Distributed Algorithms.* [Online]
Available at: https://www.cs.rit.edu/~ark/spring2013/730/module04/notes.shtml
[Accessed 2016].

Mann, J., 2015. *The Internet of Things: Opportunities and Applications across Industries.* [Online]
Available at: http://iianalytics.com/

APENDIX A. Run_Agents.sh

```
#!/bin/bash

AGENTSNR=5
index=1
clear
echo "Run $AGENTSNR Agents!"
java jade.Boot -gui -services \
"jade.core.messaging.TopicManagementService;\
jade.core.event.NotificationService;jade.core.mobility.AgentMobilityServi
ce" -agents "Vehicle_1:Agent_TMS(55, 10, 5, 5, 25, true, 0)"&
sleep 5
while [ "$index" -lt $AGENTSNR ]
do
    #echo "$index"
    java jade.Boot -container -services \
    "jade.core.messaging.TopicManagementService;\

jade.core.event.NotificationService;jade.core.mobility.AgentMobilityServi
ce" -agents "Route_$index:Agent_TMS(0, 0, 0, 0, 0, false, $index)"&
    sleep 1
    index=`expr $index + 1`
done
```

APENDIX B. 1.1 Run_Additional_Vehicle.sh

```
#!/bin/bash

AGENTSNR=5
index=1
clear
echo "Run $AGENTSNR Agents!"
java jade.Boot -container -services \
"jade.core.messaging.TopicManagementService;\
jade.core.event.NotificationService;jade.core.mobility.AgentMobilityServi
ce" -agents "Vehicle_2:Agent_TMS(5, 5, 80, 5, 5, true, 0)"&
```

APENDIX C. Agent_TMS.java

```
import jade.misc.*;
import jade.core.behaviours.TickerBehaviour;
import java.util.Random;

import jade.content.*;
import jade.content.abs.*;
import jade.content.onto.*;
import jade.content.onto.basic.*;
import jade.content.lang.*;
import jade.content.lang.sl.*;

import jade.core.Agent;
import jade.core.AID;
import jade.core.ServiceException;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.FSMBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.ParallelBehaviour;
import jade.core.behaviours.WakerBehaviour;
import jade.core.messaging.TopicManagementService;
import jade.core.messaging.TopicManagementHelper;
import jade.domain.FIPANames;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.util.Logger;
import jade.util.leap.Serializable;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAAgentManagement.DFAgentDescription;


import java.util.Date;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Collections;

public class Agent_TMS extends Agent
{
    public static final String TMS_TOPIC = "TMS-Topic";
    private AID tmsTopic;
    private LeadershipManager lead_man = new LeadershipManager();
    //Register Traffic Management System Ontology
    private ContentManager manager = (ContentManager)
getContentManager();
    private Codec codec = new SLCodec();
    private Ontology   ontology = TMSOntology.getInstance();

    /*Weights for each objective given as arguments*/
    private Byte travelTime;
    private Byte travelDistance;
    private Byte arterialStreets;
    private Byte turns;
    private Byte roadwayChanges;
```

```
    private Boolean vehicle;
    private Byte route_nr;
    private Logger myLogger = Logger.getMyLogger(getClass().getName());
    private ArrayList<Owns> routesList = new ArrayList<Owns>(); //List
for saving the routes
    static Float[] arrValues = new Float [] {0f,0f,0f,0f};
    private float g;
    private int cnt;
    //Simulate values
    public static final Float[][] SIM_VALUES = new Float [][] {
                                                                    {0.11f,
0.52f, 0.69f, 0.88f, 0.45f},
                                                                    {0.88f,
0.12f, 0.12f, 0.73f, 0.99f},
                                                                    {0.23f,
0.22f, 0.88f, 0.43f, 0.25f},
                                                                    {0.44f,
0.43f, 0.19f, 0.25f, 0.81f}
                                                                    };

    protected void setup()
    {
        Object[] args = getArguments();
     /*
     Arguments:
     Travel Time (s)         -> travelTime
     Travel Distance (km)    -> travelDistance
     Arterial Streets    -> arterialStreets
     Number of turns         -> turns
     Roadway clas. Changes   -> roadwayChanges
     Vehicle -> is it a vehicle?
     */
        if (args.length > 0)
        {
            try {
             travelTime          = Byte.parseByte((String) args[0]);
             travelDistance      = Byte.parseByte((String) args[1]);
             arterialStreets     = Byte.parseByte((String) args[2]);
             turns               = Byte.parseByte((String) args[3]);
             roadwayChanges      = Byte.parseByte((String) args[4]);
             vehicle             = Boolean.parseBoolean((String) args[5]);
             route_nr            = Byte.parseByte((String) args[6]);
             route_nr--;
            } catch (NumberFormatException e) {
                   System.out.println("Error in argument format!!");
            }
        }
        if (vehicle) //Print this info only if agent is a vehicle
        {
System.out.println("\n****************************************");
            System.out.println("Agent:                "+ getLocalName());
            System.out.println("Travel Time:          "+ travelTime + "%");
            System.out.println("Travel Distance:      "+ travelDistance +
"%");
            System.out.println("Arterial Streets:     "+ arterialStreets +
"%");
```

```
            System.out.println("Turns:              "+ turns + "%");
            System.out.println("Roadway Changes:    "+ roadwayChanges +
"%");
            System.out.println("****************************************"
+ "\n");
        }
        lead_man.setProposalResponseTimeout(3000);
        lead_man.setProposalRetryInterval(4000);

        manager.registerLanguage(codec);
        manager.registerOntology(ontology);

    try {
        TopicManagementHelper topicHelper = (TopicManagementHelper)
getHelper(TopicManagementHelper.SERVICE_NAME);
        tmsTopic = topicHelper.createTopic(TMS_TOPIC);
        topicHelper.register(tmsTopic);
    }
    catch (Exception e) {
            System.err.println("Agent "+getLocalName()+": ERROR creating
topic \"JADE\"");
            e.printStackTrace();
        }
        if(vehicle) //If Agent is a vehicle, handle InformBehaviour
        {
            addBehaviour(new HandleInformBehaviour(this));
            addBehaviour(new TickerBehaviour(this, 10000) {
                public void onTick() {
                    TMS myTMS = new TMS();
                    myTMS.setName("All");
                    routesList.clear();
                    addBehaviour(new QueryConditionsBehaviour(myAgent,
myTMS));
                }
            } );
        }
        else //If Agent is infrastructure, prepare for sending variables
        {
            addBehaviour(new HandleQueryBehaviour(this));
            TMS myTMS = new TMS();
            myTMS.setTravelTime(0.0f);
            myTMS.setTravelDistance(0.0f);
            myTMS.setName(getLocalName());
            addBehaviour(new InformOwnsBehaviour(this, myTMS));
            addBehaviour(new TickerBehaviour(this, 10000) {
                public void onTick() {
                    //g = (float) Math.random();
                    cnt++;
                    if(cnt==5)
                        cnt=0;
                        for(int i =0; i<4; i++)
                        {
                            arrValues[i] = SIM_VALUES[i][cnt];
                        }
                }
            } );
        }
```

```
        try {
            lead_man.init(this);
        }
        catch (Exception e) {
            // Should never happen
            e.printStackTrace();
        }
        try {
            addBehaviour(new TickerBehaviour(this, 5000) {
                private int nleads = 0;
                public void onTick() {
                    if ( ((Agent_TMS)myAgent).lead_man.isLeader() )
                    {
                        nleads++;
                    }
                }
            } );



        }
        catch (Exception e) {
            System.err.println("Agent "+getLocalName()+": ERROR creating
topic \"JADE\"");
            e.printStackTrace();
        }
        try {
            // Periodically send messages about topic "JADE"
            addBehaviour(new TickerBehaviour(this, 5000) {
                public void onTick() {
                    lead_man.updateLeadership();
                }
            } );
        }
        catch (Exception e) {
            System.err.println("Agent "+getLocalName()+": ERROR creating
topic \"JADE\"");
            e.printStackTrace();
        }
    }

    class InformOwnsBehaviour extends OneShotBehaviour {
            private TMS trafficMS;
            public InformOwnsBehaviour(Agent a, TMS trafficMS) {
                super(a);
                this.trafficMS = trafficMS;
            }

            public void action() {
            try {

System.out.println("\n********************************");
                    System.out.println("Agent: " + trafficMS.getName());
                    System.out.println("Travel Time:
"+trafficMS.getTravelTime());
```

```
                    System.out.println("Travel Distance:
"+trafficMS.getTravelDistance());
                    System.out.println("Arterial Streets:
"+trafficMS.getArterialSt());
                    System.out.println("Number of Turns:
"+trafficMS.getNumberOfTurns());
                    System.out.println("Roadway Changes:
"+trafficMS.getRoadwayChanges());

System.out.println("*********************************");
                    // Prepare the message
                    ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                    msg.setSender(getAID());
                    msg.addReceiver(tmsTopic);
                    msg.setLanguage(codec.getName());
                    msg.setOntology(ontology.getName());

                    // Fill the content
                    Owns owns = new Owns();
                    owns.setOwner(getAID());
                    owns.setEntityTMS(trafficMS);

                    manager.fillContent(msg, owns);
                    send(msg);
            }
            catch(Exception e) {
                e.printStackTrace();
            }

            }
    } //End Inform Owns behavior

    class HandleInformBehaviour extends CyclicBehaviour {

            public HandleInformBehaviour(Agent a) {
                super(a);
            }

            public void action() {
                ACLMessage msg =
receive(MessageTemplate.and(MessageTemplate.MatchTopic(tmsTopic),
MessageTemplate.MatchPerformative(ACLMessage.INFORM)));//receive(MessageT
emplate.MatchPerformative(ACLMessage.INFORM));
                if (msg != null) {
                    //System.out.println("\nVEHICLE: Information received
from ROUTE. Message is ");
                    //System.out.println(msg);
                try {
                        ContentElement ce = manager.extractContent(msg);
                        if (ce instanceof Owns) {
                            Owns owns = (Owns) ce;
                            AID owner = owns.getOwner();
                            TMS myTMS = owns.getEntityTMS();
                            int indexfound = -1;
                            System.out.println("Received: " +
myTMS.getName());
                            for (int u = 0; u<routesList.size(); u++)
```

```
                                {
                                    //System.out.println("+++ " +
routesList.get(u).getEntityTMS().getName() + " == "+ myTMS.getName() +"?
list size: " + routesList.size());

if(myTMS.getName().equals(routesList.get(u).getEntityTMS().getName()))
                                    {
                                        indexfound = u;
                                        break;
                                    }
                                }

                                if(indexfound == -1)
                                {
                                    routesList.add(owns);
                                    System.out.println("Add: " +
owns.getEntityTMS().getName());
                                }
                                else
                                {
                                    routesList.set(indexfound, owns);
                                }
                                Owns bestRouteP = new Owns();
                                TMS bestRouteObj = new TMS();
                                bestRouteP = calculateBestRoute(routesList);
                                bestRouteObj = bestRouteP.getEntityTMS();
                                System.out.println("*-*-*->> Best Route: " +
bestRouteObj.getName());
                            }
                    else {
                            System.out.println("Unknown predicate
"+ce.getClass().getName());
                        }
                }

                catch(Exception e) {
                    e.printStackTrace();
                }
            }
            else {
                block();
            }
            }

            private Owns calculateBestRoute (ArrayList<Owns> routesList)
            {
                Iterator i = routesList.iterator();
                Owns owns = new Owns();
                TMS myTMS = new TMS();
                float sumUtility = 0.0f;
                ArrayList<Float> arrUtilities = new ArrayList<Float>(4);
                float calcRoadwayChanges = 0.0f;
                while (i.hasNext())
                {
                    owns = (Owns) i.next();
                    myTMS = owns.getEntityTMS();
```

```
                    sumUtility = myTMS.getTravelTime() *
(((Agent_TMS)myAgent).travelTime * 0.01f); //Utility = N*Weight
                    sumUtility = sumUtility + myTMS.getTravelDistance() *
(((Agent_TMS)myAgent).travelDistance * 0.01f); //Utility = N*Weight
                    sumUtility = sumUtility + myTMS.getArterialSt() *
(((Agent_TMS)myAgent).arterialStreets * 0.01f); //Utility = N*Weight
                    sumUtility = sumUtility + myTMS.getNumberOfTurns() *
(((Agent_TMS)myAgent).turns * 0.01f); //Utility = N*Weight
                    sumUtility = sumUtility + myTMS.getRoadwayChanges() *
(((Agent_TMS)myAgent).roadwayChanges * 0.01f); //Utility = N*Weight
                    calcRoadwayChanges = myTMS.getRoadwayChanges() *
(((Agent_TMS)myAgent).roadwayChanges * 0.01f); //Utility = N*Weight
                    arrUtilities.add(sumUtility);
                    sumUtility = 0;
                }
                System.out.println("Utilities: " + arrUtilities );
                int n = getMax(arrUtilities);
                return routesList.get(n);
            }

            public int getMax(ArrayList list)
            {
                float max = -1f;
                int n = -1;
                for(int i=0; i<list.size(); i++)
                {
                    if((float) list.get(i) > max)
                    {
                        max = (float) list.get(i);
                        n = i;
                    }
                }
                return n;
            }

        } //End of Handle Inform Behavior


    class QueryConditionsBehaviour extends OneShotBehaviour {
                TMS myTMS;

                public QueryConditionsBehaviour(Agent a, TMS rxTMS) {
                    super(a);
                    this.myTMS = rxTMS;
                }

                public void action() {
                try {
                        System.out.println("\nVehicle: Query Conditions
of "+myTMS.getName());

                        // Prepare the message
                        ACLMessage msg = new
ACLMessage(ACLMessage.QUERY_REF);
                        msg.setSender(getAID());
                        msg.addReceiver(tmsTopic);
                        msg.setLanguage(codec.getName());
```

```
                        msg.setOntology(ontology.getName());

                        // Fill the content
                        Ontology onto = TMSOntology.getInstance();
                        AbsVariable x = new AbsVariable("x",
TMSOntology.TMS);

                        AbsPredicate owns = new
AbsPredicate(TMSOntology.OWNS);
                        owns.set(TMSOntology.TMS, (AbsTerm)
onto.fromObject(myTMS));
                        owns.set(TMSOntology.OWNS_OWNER, x);
                        owns.set(TMSOntology.OWNS_ENTITY_TMS, (AbsTerm)
onto.fromObject(myTMS));

                        AbsIRE iota = new AbsIRE(SLVocabulary.IOTA);
                        iota.setVariable(x);
                        iota.setProposition(owns);

                        manager.fillContent(msg, iota);
                        send(msg);

                }
                catch(Exception e) {
                    e.printStackTrace();
                }

                }
        } //End of Query

    class HandleQueryBehaviour extends CyclicBehaviour {

            public HandleQueryBehaviour(Agent a) {
                super(a);
            }

            public void action() {
                ACLMessage msg =
receive(MessageTemplate.and(MessageTemplate.MatchPerformative(ACLMessage.
QUERY_REF), MessageTemplate.MatchTopic(tmsTopic)));
                        if (msg != null) {
                try {

                        AbsIRE ire = (AbsIRE)
manager.extractAbsContent(msg);
                        if (ire.getTypeName().equals(SLVocabulary.IOTA))
{
                            AbsPredicate p = (AbsPredicate)
ire.getProposition();
                            AbsConcept absIt = (AbsConcept)
p.getAbsTerm(TMSOntology.OWNS_ENTITY_TMS);
                            TMS TrafficMS = (TMS)
ontology.toObject(absIt);

                            TMS myTMS = new TMS();
                            myTMS.setTravelTime(SIM_VALUES[route_nr][0]);
```

```
myTMS.setTravelDistance(SIM_VALUES[route_nr][1]);
                            myTMS.setArterialSt(SIM_VALUES[route_nr][2]);

myTMS.setNumberOfTurns(SIM_VALUES[route_nr][3]);

myTMS.setRoadwayChanges(SIM_VALUES[route_nr][4]);
                            myTMS.setName(getLocalName());
                            addBehaviour(new InformOwnsBehaviour(myAgent,
myTMS));
                        }
                    }

                catch(Exception e) {
                    e.printStackTrace();
                }
            }
            else {
                block();
            }
            }

            private void getConditionValues (ArrayList<Float> values,
byte size) //Obtain random values for conditions of Routes
            {
                float f;
                for(byte i=0; i<size; i++)
                {
                    f = (float) Math.random();
                    values.add((Float) f);
                }

            }
    } //End Handle Query Behavior


} //End Agent_TMS class

…
```

APENDIX D. Owns.java

```
import jade.content.Predicate;
import jade.core.AID;

public class Owns implements Predicate {
    private static final long serialVersionUID = 1L;

    private AID owner;
    private TMS entityTMS;

    public AID getOwner() {
        return owner;
    }

    public void setOwner(AID id) {
        owner = id;
    }

    public TMS getEntityTMS() {
        return entityTMS;
    }

    public void setEntityTMS(TMS t) {
        entityTMS = t;
    }
}
```

APENDIX E. TMS.java

```
import jade.content.*;

public class TMS implements Predicate
{
          private String name;
  private float travelTime;
          private float travelDistance;
          private float arterialSt;
          private float numberOfTurns;
          private float roadwayChanges;

          public String getName()
          {
                    return name;
          }

          public void setName (String name)
          {
                    this.name = name;
          }

          public float getTravelTime()
          {
                                        return travelTime;
    }
          public void setTravelTime(float travelTimeArg)
          {
                                        travelTime = travelTimeArg;
    }

          public float getTravelDistance()
          {
                                        return travelDistance;
    }
          public void setTravelDistance(float travelDistanceArg)
          {
                                        travelDistance = travelDistanceArg;
    }

          public float getArterialSt()
          {
                              return arterialSt;
          }
          public void setArterialSt(float arterialStArg)
          {
                              arterialSt = arterialStArg;
          }

          public float getNumberOfTurns()
          {
                              return numberOfTurns;
          }
          public void setNumberOfTurns(float numberOfTurnsArg)
          {
                              numberOfTurns = numberOfTurnsArg;
          }
```

```
	public float getRoadwayChanges()
	{
			return roadwayChanges;
	}
	public void setRoadwayChanges(float roadwayChangesArg)
	{
			roadwayChanges = roadwayChangesArg;
	}

}
```

APENDIX F. TMSOntology.java

```
import jade.content.onto.*;
import jade.content.schema.*;
import jade.content.schema.facets.*;

public class TMSOntology extends Ontology implements TMSVocabulary {
    // The name identifying this ontology
    public static final String ONTOLOGY_NAME = "TMSOntology";

    // ----------> The singleton instance of this ontology
    private static Ontology theInstance = new
TMSOntology(BasicOntology.getInstance());
    // ----------> Method to access the singleton ontology object
    public static Ontology getInstance()
    {
        return theInstance;
    }

    // Private constructor
    private TMSOntology(Ontology base) {

        super(ONTOLOGY_NAME, base);

        try
        {
            add(new PredicateSchema(OWNS), Owns.class);
            add(new ConceptSchema(TMS), TMS.class);
            ConceptSchema cs = (ConceptSchema) getSchema(TMS);
            cs.add(TMS_NAME, (PrimitiveSchema)
getSchema(BasicOntology.STRING));
            cs.add(TMS_TRAVELTIME, (PrimitiveSchema)
getSchema(BasicOntology.FLOAT), ObjectSchema.OPTIONAL);
            cs.add(TMS_TRAVELDISTANCE, (PrimitiveSchema)
getSchema(BasicOntology.FLOAT), ObjectSchema.OPTIONAL);
            cs.add(TMS_ARTERIAL_ST, (PrimitiveSchema)
getSchema(BasicOntology.FLOAT), ObjectSchema.OPTIONAL);
            cs.add(TMS_NUMBEROFTURNS, (PrimitiveSchema)
getSchema(BasicOntology.FLOAT), ObjectSchema.OPTIONAL);
            cs.add(TMS_ROADWAYCHANGES, (PrimitiveSchema)
getSchema(BasicOntology.FLOAT), ObjectSchema.OPTIONAL);

            PredicateSchema ps = (PredicateSchema) getSchema(OWNS);
            ps.add(OWNS_OWNER, (ConceptSchema)
getSchema(BasicOntology.AID), ObjectSchema.OPTIONAL);
            ps.add(OWNS_ENTITY_TMS, (ConceptSchema) getSchema(TMS));
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

APENDIX G. TMS_Vocabulary.java

```
public interface TMSVocabulary  {
  public static final String TMS = "TMS";
  public static final String TMS_TRAVELTIME = "travelTime";
  public static final String TMS_TRAVELDISTANCE = "travelDistance";
  public static final String TMS_ARTERIAL_ST = "arterialSt";
  public static final String TMS_NUMBEROFTURNS = "numberOfTurns";
  public static final String TMS_ROADWAYCHANGES = "roadwayChanges";
  public static final String TMS_NAME = "Name";

  public static final String OWNS = "OWNS";
  public static final String OWNS_OWNER = "Owner";
  public static final String OWNS_ENTITY_TMS = "entityTMS";
}
```

APENDIX H. Screenshots

```
miguel@satellite: ~/jade_temp/tsd
*********************************
Agent: Route_2
Travel Time: 0.88
Travel Distance: 0.12

Arterial Streets: 0.12
Number of Turns: 0.73
Roadway Changes: 0.99
*********************************
Utilities: [0.27650002]
*-*-*->> Best Route: Route_3
Received: Route_1
Add: Route_1
Utilities: [0.27650002, 0.3035]
*-*-*->> Best Route: Route_1
Received: Route_4
Add: Route_4
Utilities: [0.27650002, 0.3035, 0.50949997]
*-*-*->> Best Route: Route_4
Received: Route_2
Add: Route_2
Utilities: [0.27650002, 0.3035, 0.50949997, 0.78599995]
*-*-*->> Best Route: Route_2
```

Figure 12. Running Agents

```
miguel@satellite: ~/jade_temp/tsd
INFO: Service jade.core.resource.ResourceManagement initialized
Nov 13, 2016 6:15:10 PM jade.core.BaseService init
INFO: Service jade.core.messaging.TopicManagement initialized
Nov 13, 2016 6:15:10 PM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
Nov 13, 2016 6:15:10 PM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized

****************************************
Agent:              Vehicle_2
Travel Time:        5%
Travel Distance:    5%
Arterial Streets:   80%
Turns:              5%
Roadway Changes:    5%
****************************************

Nov 13, 2016 6:15:11 PM jade.core.AgentContainerImpl startBootstrapAgents
INFO: Container-Monitor agent activated
Nov 13, 2016 6:15:11 PM jade.core.AgentContainerImpl joinPlatform
INFO: --------------------------------------
Agent container Container-5@192.168.15.4 is ready.
--------------------------------------------
```

Figure 13. Vehicle Weights

```
miguel@satellite: ~/jade_temp/tsd
*-*-*->> Best Route: Route_3
Received: Route_1
Add: Route_1
Utilities: [0.7604999, 0.24849999, 0.232, 0.6499999]
*-*-*->> Best Route: Route_3

Vehicle: Query Conditions of All
Received: Route_2
Add: Route_2
Utilities: [0.232]
*-*-*->> Best Route: Route_2
Received: Route_4
Add: Route_4
Utilities: [0.232, 0.24849999]
*-*-*->> Best Route: Route_4
Received: Route_1
Add: Route_1
Utilities: [0.232, 0.24849999, 0.6499999]
*-*-*->> Best Route: Route_1
Received: Route_3
Add: Route_3
Utilities: [0.232, 0.24849999, 0.6499999, 0.7604999]
*-*-*->> Best Route: Route_3
```

Figure 14. Running Vehicle 2

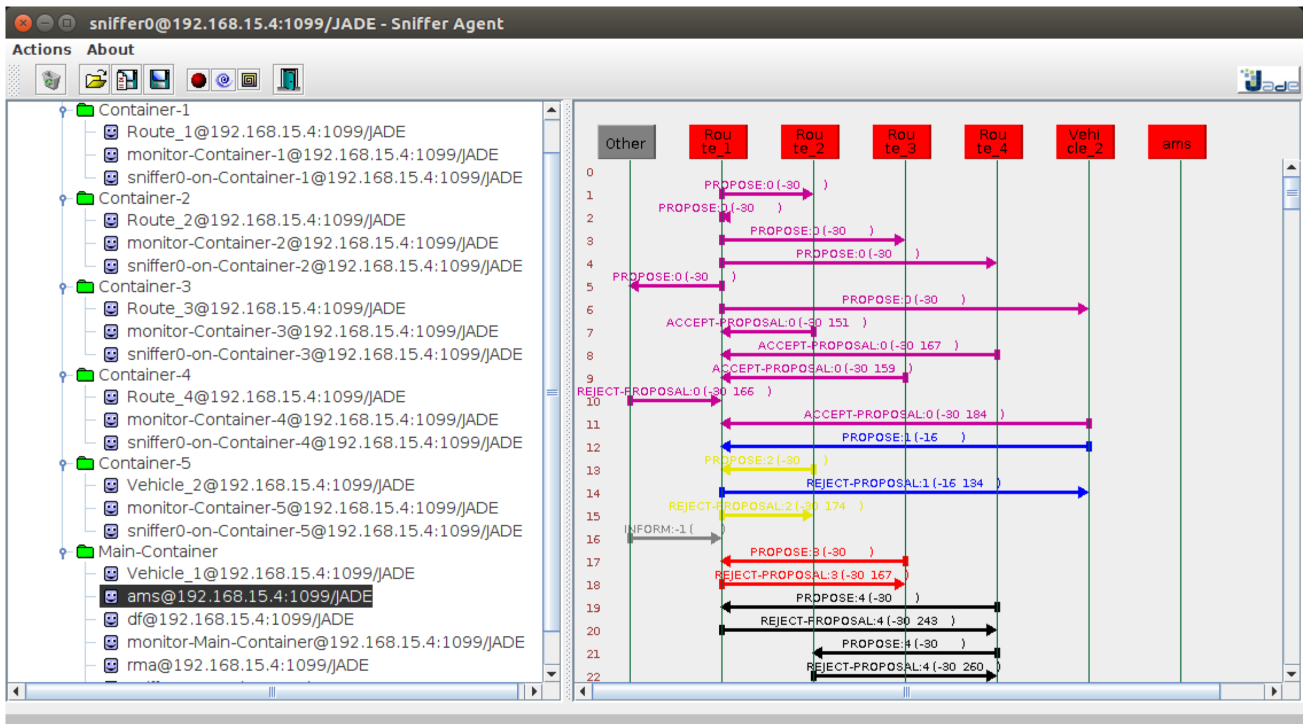

Figure 15. Example of agents' communication