Emmanuel Katende Dinanga

MULTI AGENT SYSTEMS APPROACH FOR EMERGENCY RESPONSE PROCESS MANAGEMENT

Dissertation presented to the Graduate Program in Automation and Systems Engineering in partial fulfillment of the requirements for the degree of Master in Automation and Systems Engineering.

Advisor: Prof. Dr. Jomi F. Hubner
Co-advisor: Prof. Dr. Rodrigo C. Carlson

Florianópolis

2016

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Katende, Emmanuel Dinanga
MULTI AGENT SYSTEMS APPROACH FOR EMERGENCY RESPONSE PROCESS MANAGEMENT / Emmanuel Dinanga Katende ; orientadora, Jomi Fred Hubner ; coorientadora, Rodrigo Castelan Carlson. - Florianópolis, SC, 2016.
152 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Inclui referências

1. Engenharia de Automação e Sistemas. 2. Sistemas Multi Agentes. 3. Controle de Tráfego Urbano. 4. Resposta a Emergência. I. Hubner, Jomi Fred. II. Carlson, Rodrigo Castelan. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. IV. Título.

Emmanuel Katende Dinanga

MULTI AGENT SYSTEMS APPROACH FOR EMERGENCY RESPONSE PROCESS MANAGEMENT

This Dissertation is recommended in partial fulfillment of the requirements for the degree of "Master in Automation and Systems Engineering", which has been approved in its present form by the Graduate Program in Automation and Systems Engineering.

Florianópolis, September 28th 2016.

Prof. Dr. Rômulo S. de Oliveira
Graduate Program Coordinator
Federal University of Santa Catarina

Dissertation Committee:

Prof. Dr. Jomi F. Hubner
Advisor
Federal University of Santa Catarina

Prof. Dr. Rodrigo C. Carlson
Co-advisor
Federal University of Santa Catarina

Prof. Dra. Jerusa Marchi
Federal University of Santa Catarina

Prof. Dr. Ubirajara F. Moreno
Federal University of Santa Catarina

Prof. Dr. Werner Kraus Jr.
Federal University of Santa Catarina

ACKNOWLEDGEMENTS

I would first like to thank my dissertation advisor Dr. Jomi F. Hubner. The door to Prof. Jomi office was always open and his e-mail always available, whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this dissertation to be my own work, but steered me in the right direction whenever he thought I needed it.

I would also like to thank my co-advisor Dr. Rodrigo C. Carlson who provided his great knowledge and powerful experience in urban traffic domain, for the validation survey and realization of this dissertation. Without his passionate participation and input, this work would not have been successfully conducted.

I would also like to thank my MAS lab-mates for the stimulating discussions around my research topic, the exchange of knowledge and any other kind of help during the realization of my dissertation.

I would also like to acknowledge Dr. Werner Kraus Jr., Dra. Jerusa Marchi and Dr. Ubirajara F. Moreno, to have accepted to be part of the jury in my defense.

These years of studies and research would have been almost impossible without the support of the Coordination for the Improvement of Higher Education Personnel (CAPES).

I must express my very profound gratitude to my spouse, my parents and to all my family for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this dissertation. This accomplishment would not have been possible without them. Thank you.

Finally, I thank PGEAS, UFSC, Brazil and all who closely as far, participated, prayed and cheered for me to get this title which this work is part of partial evaluation. I will always be grateful to you all. Thank you very much!

For I am certain that not death, or life, or angels, or rulers, or things present, or things to come, or powers, or things on high, or things under the earth, or anything which is made, will be able to come between us and the love of God which is in Christ Jesus our Lord

Romans 8:38-39

RESUMO

Esta dissertação aborda o problema de melhorar o processo de resposta a uma emergência em um sistema de trânsito urbano. A abordagem de Sistemas Multi Agentes, dotado de uma técnica de cooperação, é adotada para implementar uma estratégia que controla os semáforos e conduz os veículos de emergências de modo a melhorar seus tempos de deslocamento, além de minimizar o impacto das prioridades atribuídas a tais veículos no fluxo do tráfego. A proposta é avaliada definindo como métricas: o tempo total necessário para resolver a emergência, como também, a velocidade média, o tempo médio e a densidade média de todos os veículos no trânsito. Os resultados mostram que nossa proposta consegue reduzir o tempo de deslocamento dos veículos de emergência, além de minimizar o impacto das prioridades atribuídas a tais veículos no fluxo do tráfego.

Palavras-chave: Sistemas Multi Agentes, Controle de Tráfego Urbano, Processo de Resposta a Emergência

ABSTRACT

This dissertation approaches the problem of improving an emergency response process on an urban traffic system. The use of Multi Agent Systems approach, endowed with an explicit cooperation technique is proposed to implement a strategy that controls the traffic signals and route emergency vehicles in order to improve their travel time and minimize the impact of priorities given to these emergency vehicles on the traffic flow. The time needed to perform all the emergency response process, as well as, the average speed, travel time and density are defined as metrics for the assessment. The assessment results show that our proposal is able to reduce the travel time of emergency vehicles as well as to minimize the impact of priorities given to emergency vehicles on the traffic flow.

Keywords: Multi Agent Systems, Urban Traffic Control, Emergency Response Process

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Preventing road traffic injuries from occurring should be the main goal to be pursued, but the reality is that crashes continue to occur. Society therefore has to be prepared to mitigate the consequences of crashes and enhance the quality of life of people who are injured. The way in which persons injured in emergencies are dealt with following a crash determines their chances and the quality of survival. (Mohan 2006).

The growth of the economy, industry and population has turned urban areas increasingly dense and interconnected, due to numerous reasons and needs such as habitation, work, transportation and even entertainment. As a consequence, these areas became vulnerable to several kinds of emergencies such as traffic accidents, building fires, gas leaks and unexpected health crisis situations (Monares et al. 2012). For instance, road traffic injuries are the leading cause of death among young people from 15 to 29 years; and 90% of the world's fatalities on the roads occur in low and middle-income countries (WHO 2015), which probably do not have advanced and effective post crash care systems (WHO 2015).

1.1 MOTIVATION

Emergencies are often managed by a process that starts when an urban emergency center receives the notification of an incident classified as an emergency and ends when such situation is recovered. This process requires fast and effective responses from emergency teams such as: firefighters, medical personnel, and police officers.

Borges et al. consider three main stages that compose an emergency management process: a *notification and validation*, which starts when an incident notification is reported, a *response process*, which is the focus of this dissertation and takes place once the crisis is verified as true, and a *closing activity*, which involves basically reporting the result of the response process.

In Brazil, the Urban Mobile Aid Service (SAMU) is one of the emergency teams usually involved in emergency response processes. In the State of Santa Catarina, Brazil, the SAMU registered 956,297 emergency calls and assisted 325,129 in 2015. Among the assisted persons, 143,445 (44%) required ambulance interventions and unfortunately, 3020 (1%) of them died (SAMU 2015). In 2014, the SAMU's statistics reported that from cases registered as death, 70%

occurred before the emergency team arrives, 27% during the care and 3% during the transportation to health care locations.

Although the main goal must be preventing emergencies to occur, the pre-hospital time-frame, i.e., the time between the crash and access to emergency medical care is critical for successful patient management. McCoy et al. analyzed 19,167 trauma patients from which 865 (4.5%) died. Among these patients, 16,170 (84%) injuries were blunt, with 596 (3.7%) deaths; and 2,997 (16%) injuries were penetrating, with 269 (9%) deaths. They analyzed the relationship of scene time and transport time with mortality and observed that a scene time grater than 20 minutes was associated with higher odds of mortality than scene time less then 10 minutes. Thus, access to pre-hospital services and quick evacuation and transportation to hospital can save many lives, since the majority of deaths occurs before reaching the hospital (WHO 2013).

There are various factors that can delay the assistance of victims during an emergency response process. Some of these factors are: (1) the difficulty of emergency vehicles to reach both the incident location and health care locations, due urban traffic congestions or disturbing events occurring in the urban area involved; (2) a bad (or a lack of) cooperation among several services involved in the emergency situation, as well as a poor (or lack of) coordination of their activities or/and resources.

Regarding the difficulty of emergency vehicles displacement which may refer basically to a traffic control problem, a considerable number of advanced solutions have been proposed in the literature in order to cope with urban traffic congestions and similar issues. Such solutions are discussed and researched with more emphasis in domains like Adaptive Control Theory, (Liu et al. 2007), Algorithms and Graphs, (Kang et al. 2013), Artificial Intelligence, (Shumin et al. 2010), Optimization, (Wu et al. 2010), etc.

Concerning the difficulty of cooperation and coordination during an emergency response process, numerous researches have been conducted in several domains in the literature, from which we highlight Multi Agent Systems (MAS), in which the coordination is considered as the most researched topic in the area (Wooldridge 2009). Some of the works published in the MAS field regarding cooperation for urban traffic issues are: Au et al., Vilarinho et al., Rodríguez et al., Dresner and Stone, Tomás and Garcia.

Considering the two difficulties presented regarding the emergency response process, this dissertation answers the following question: can we use the MAS approach with an explicit cooperation

technique and coordination mechanism to implement a strategy in order to improve an emergency response process?. More specifically, we focus on the traffic lights control and the emergency vehicles management and routing during an emergency response process presented in Section 2.3. We propose the use of a MAS approach endowed with an emergency response strategy that controls the traffic lights and manages emergency vehicles in order to reach the objectives established in Section 1.2.

1.2 AIM

This dissertation aims to propose a multi agent system model endowed with an explicit cooperation technique and coordination mechanism, which implements some strategy that controls an urban traffic system, as well as managing and routing emergency teams in order to improve an emergency response process. The objectives of such aim are:

1. Reduce the travel time of emergency teams during the displacement from their bases to the emergency location and than to health care locations.
2. Reduce the impact of priorities assigned to emergency teams on the traffic flow.

1.3 TEXT STRUCTURE

This dissertation is divided into five chapters: Chapter 2 introduce the main concepts with respect to Multi Agent Systems, Traffic Control and Emergency Management. Chapter **??** reports the state of the art regarding the traffic control under an emergency response process using both MAS and others approaches. In Chapter 3, we present our solution method comprising basically an emergency response strategy and a MAS model, as well as, we summarizes the implementation of our proposal. In Chapter 4, we assess our proposal, show the results according to some metrics defined therein. Lastly, we conclude our dissertation in Chapter 5.

2 LITERATURE REVIEW

The Multi Agent System field is highly interdisciplinary: it takes inspiration from such diverse areas as economics, philosophy, logic, ecology, and the social sciences. It should come as no surprise that there are therefore many different views of what the Multi Agent System project is all about. (Wooldridge 2009)

2.1 MULTI AGENT SYSTEMS

Most of researchers consider and present Multi Agent Systems (MAS) as a society of agents. Ferber et al. 2004 present MAS as a set of agents that interact together to coordinate their behavior and often cooperate to achieve some collective goal. Wooldridge 2009 and Malowanczyk 2014 present MAS in the sense of a computer system. Wooldridge 2009, in particular, presents MAS as a number of agents interacting with one another, typically by exchanging messages through some computer network infrastructure.

Many approaches about MAS abstraction and representation have been proposed, each one focusing on a different view of MAS. One of the approaches frequently cited in MAS literature, that includes all the MAS aspects presented in the definitions cited above is *Voyelles*, proposed by Demazeau 1995.

Voyelles is an high and abstract approach that decomposes a MAS into four dimensions or levels: *Agent*, *Environment*, *Interaction*, and *Organization*. In his work, Demazeau 1995 considers that to build a MAS, the user needs to choose the agent, environment, interaction and organization models to be instantiated. He introduces *the three statements* that conceive a MAS from three points of view as follow:

- The declarative equation:

  $$MAS = Agents + Environment + Interaction + Organization;$$

- The functional equation:

  $$Function(MAS) = \sum Function(Agents) + Collective\ Function;$$

- The recursion principle:

  A MAS has to be considered as an agent at a higher level;

All the elements in these three statements are explained next in more detail. Figure 2.1 shows all the *Voyelles* levels. Each level is introduced in the sequence.

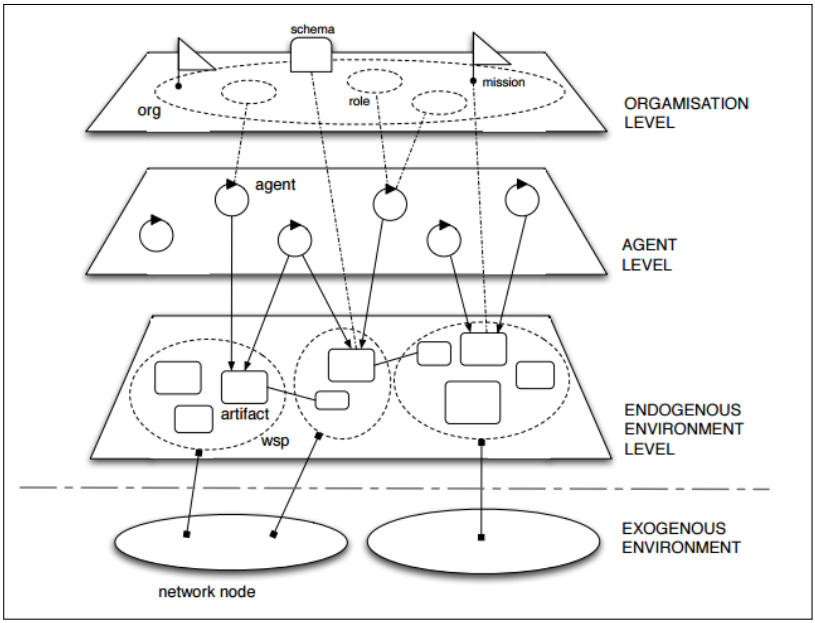

Figure 2.1: MAS levels, (Hubner Rafael Heitor Bordini 2015).

2.1.1 Agent

The Agent notion is part of the fundamental definitions that are indispensable for understanding MAS paradigm. As classified in the *Voyelles* approach, we conceive an agent as the first element of a MAS, since it defines *who* acts and interacts into such MAS. Figure 2.1 illustrates agents in the agent level, according to the *Voyelles* approach.

Various authors define agents in the MAS paradigm according to their specific domain of research; or from the way they conceive and present the MAS approach. One of the definitions that summarizes the notion of agent in the MAS paradigm considers an agent as a computer system that is situated in some environment, and that is capable of autonomous actions in this environment in order to meet its delegated objectives (Jennings 2000) and (Wooldridge 2009).

Wooldridge 2009 also affirms that an agent is capable of independent actions on behalf of its user or owner. In other words, an agent can figure out for itself what it needs to do in order to satisfy its design objectives, rather than having to be told explicitly what to do at any given moment. Luck et al. 2005 also define an agent as a computer system that is capable of flexible autonomous action in dynamic, unpredictable, typically multi-agent domains.

All these definitions mention that an agent acts in an environment without describing what is such environment. They also refer to autonomous actions and objectives without specifying such actions or objectives. This leads us to induce that agents, the environment, actions and objectives are elements that depend on the scenario in which the MAS is applied, as well as the context or role of the agents in such MAS. Therefore, agent theory can be applied in every domain in which it is possible to define or identify an environment, autonomous actions, and objectives.

2.1.2 Environment

As classified in the *Voyelles* approach, we conceive an environment as the second element in a MAS, because it defines *where* agents act and interact. The environment often has some artifacts that agents use to realize actions. When the environment is defined inside of the MAS, it is referred as an the *endogenous* environment. Figure 2.1 illustrates an endogenous environment with some artifacts in the environment level, according to the *Voyelles* approach.

Nevertheless, agents can act and interact with a system outside the MAS architecture, in which there are some artifacts through which they can realize some actions. In this case we refer to an *exogenous* environment in relation to the MAS. Figure 2.1 illustrates an exogenous environment in the environment level, according to the *Voyelles* approach.

Weyns et al. 2007 define the environment as a first-class abstraction in MAS with a dual role that provides: (1) the surrounding conditions for agents to exist, and (2) an exploitable design abstraction to build MAS applications. They consider that all non-agent elements of a MAS are typically considered to be part of the MAS environment.

Ricci et al. 2011 present the environment as a computational or physical place where agents are situated, and providing the basic ground for defining the notions of agent perception, action and interaction. They conceive the environment according to two main

perspectives. The first conception is the classical perspective rooted in Artificial Intelligence (AI), which identifies the environment as an external world. The second conception is the more recent perspective from the context of Agent-Oriented Software Engineering (AOSE) (Bernon et al. 2005, Jennings 1999, Wooldridge and Ciancarini 1999), which introduces the environment as a suitable place where to encapsulate functionalities and services to support agents activities.

2.1.3 Organization

As classified in the *Voyelles* approach, we consider the organization as the third element of a MAS, because it defines *How Agents* act and interact in a MAS *Environment*. Figure 2.1 illustrates the organization in the organization level, according to the *Voyelles* approach.

Gasser 1992 defines an organization as one that provides a framework for activity and interaction through the definition of roles, behavioral expectations and authority relationships (e. g. control). Wooldridge et al. 2000 view an organization as a collection of roles, that stand in certain relationships to one another, and that take part in systematic institutionalized patterns of interactions with other roles. Ferber et al. 2004 also mention these two definitions and derive from them what they call *the main features of organization*, which are:

1. An organization is constituted of agents (individuals) that manifest a behavior;

2. The overall organization may be partitioned into partitions that may overlap;

3. Behaviors of agents are functionally related to the overall organization activity (concept of role);

4. Agents are engaged into dynamic relationship, also called patterns of activities, which may be typed using a taxonomy of roles, tasks or protocols, thus describing a kind of supra individuality;

5. Types of behaviors are related through relationships between roles, tasks and protocols.

2.1.4 Interaction

Perhaps interactions can be considered as the motor of a dynamic MAS, since an adequate dynamic in MAS requires an adequate interaction among agents therein. The interaction consists to the way agents communicate among them as well as with the environments around them. It also refers to the behavior of these agents toward such communications. That is, how agents react with respect to a communication from other agents or from the environment in which they act.

Gomez-Sanz and Pavon 2006 define interactions in MAS as what determine the behavior of agents by showing what is their expected reaction when, for instance, receiving or sending a message or signal, or by putting an item in a shared workspace.

Interactions between agents can be *direct* or *indirect*. A *direct* interaction consists, for instance, to agents A and B exchanging messages between them without the need of an intermediary. An interaction is *indirect* when, for example, messages exchanged between the agents A and B are in different languages and have to transit through a translator in order to be traduced. Figures 2.2 and 2.3 show examples of a direct and an indirect interaction, respectively.

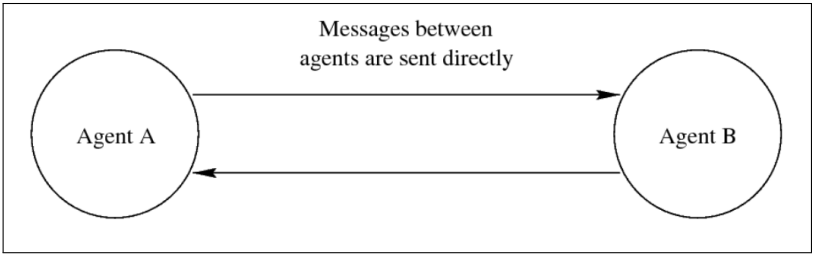

Figure 2.2: Example of a Direct Interaction.

Moreover, communications in interactions can be *synchronous* or *asynchronous*. A communication is *synchronous* when the agent execution is blocked and waits for the reply of a message before proceeding with its execution. For example: "An agent asks the price of a product in order to proceed with the payment." Such agent will not proceed with the payment until it receives the asked price or a until a timeout if considered. A communication is *asynchronous* when the agent does not wait for the reply of a message to proceed

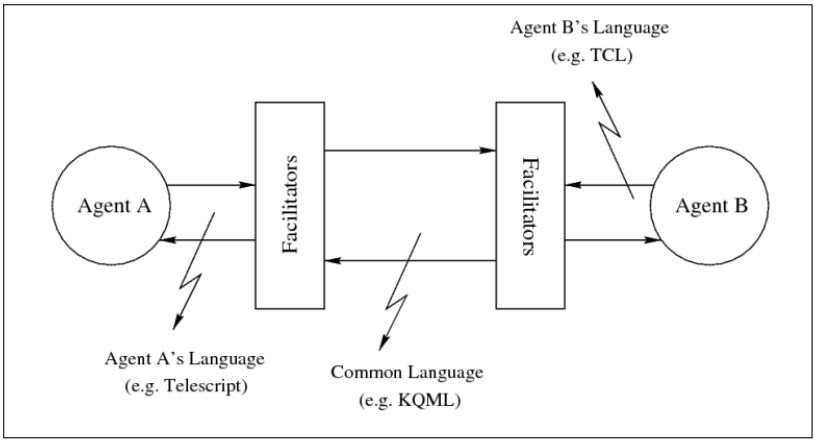

Figure 2.3: Example of an Indirect Interaction.

with the execution. For example: "An agent that asks the current time while it is walking in some place."

2.1.5 Interaction Protocol

Interactions among agents are often specified by interaction protocols. These protocols are generally used depending on the interaction context in the MAS, and usually carry with them some pattern and language that govern the exchange of messages among agents. The *Request Protocol* (FIPA Request Interaction Protocol Specification 2002) and the *Contract Net Protocol* (FIPA Contract Net Interaction Protocol Specification 2002) are examples of interaction protocols, created by The Foundation for Intelligent Physical Agent (FIPA 2002). These protocols use FIPA Agent Communication Language (ACL) (FIPA Agent Communication Language Specifications 2002) as the communication language. Figures 2.4 and 2.5 show the *Contract Net* and *Request protocols* sequence diagrams respectively.

Figure 2.4 shows the *Contract Net Protocol* execution pattern among an $initiator$ agent and m $participant$ agents. The flow begins when the $initiator$ sends m *Calls For Proposal* (cfp) to the participants. After a deadline, n participants, $n < m$, return an answer back. A number i of participants can $refuse$ the call and j participants, $j = n - i$, can $propose$ to the initiator. After an evaluation, the initiator can $reject$ k $proposals$, $k < j$, and accept l

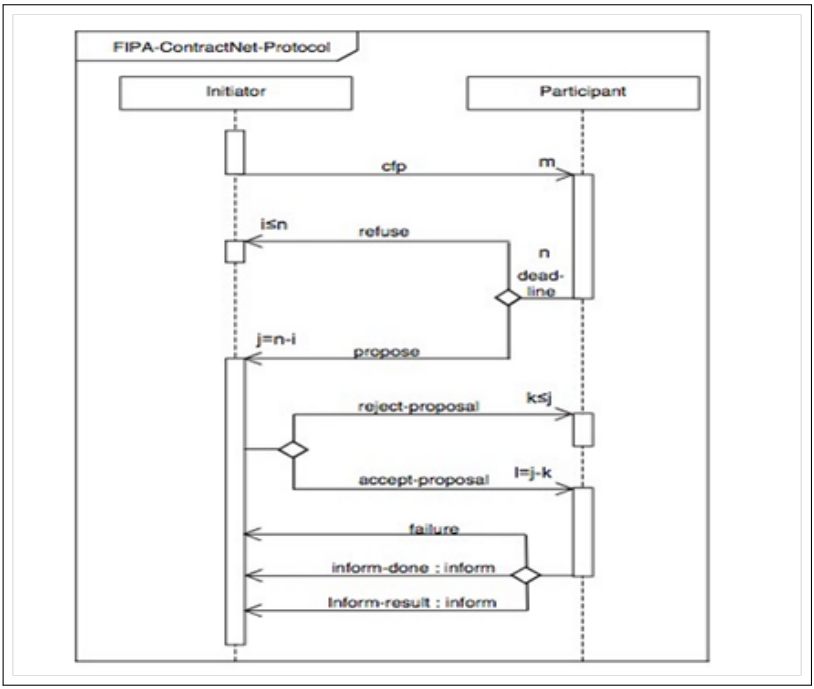

Figure 2.4: FIPA Contract Net Protocol (FIPA Contract Net Interaction Protocol Specification 2002).

$proposals$, $l = j - k$. Lastly, The l participants whose proposals were accepted send either an $inform - done$ message for acceptance, or $inform - result$ with more details, or send a $failure$ in the case of a problem.

Figure 2.5 shows the *Request Protocol* execution pattern among an $initiator$ and a $participant$. It begins when the $initiator$ sends a $request$ to the participant. Lastly, the participant that accepts the initiator request sends a message that can be either a $failure$ in the case of a problem, or an $inform - done$ message for acceptance, or also an $inform - result$ with more details about the result.

2.1.6 Cooperation

Changhong et al. 2002 consider that cooperation among agents occurs when one autonomous agent or one autonomous agent coalition adopts another autonomous agent or another autonomous agent

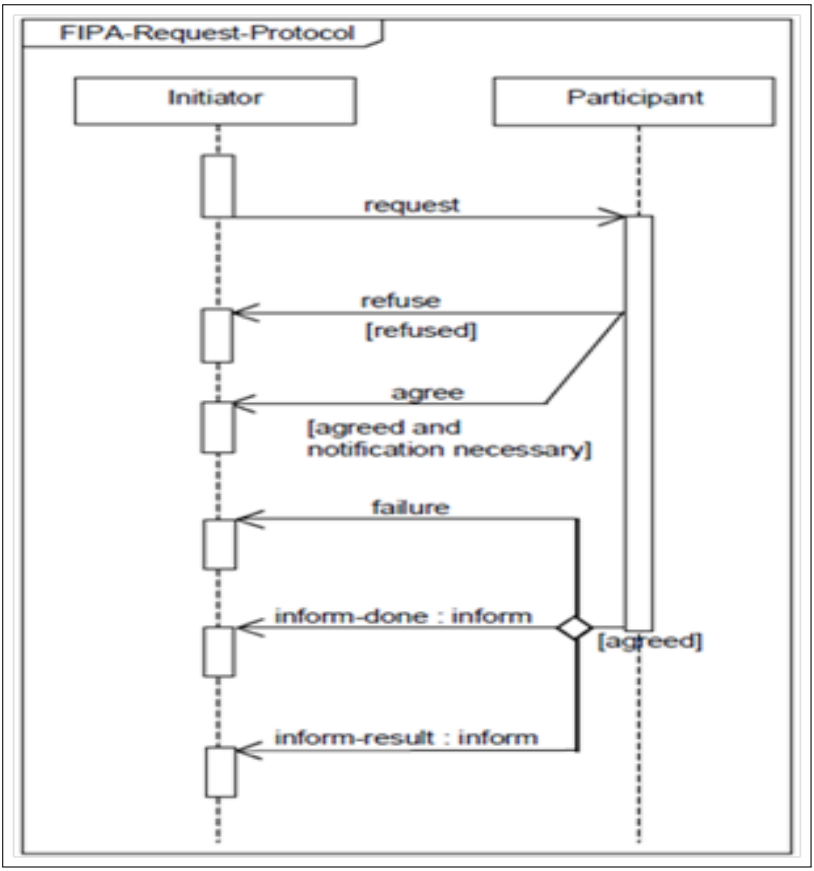

Figure 2.5: FIPA Request Protocol, (FIPA Request Interaction Protocol Specification 2002).

coalition. Wooldridge 2009 presents the cooperation in MAS as a derivation of the Cooperative Distributed Problem Solving (CDPS) approach. CDPS studies how a loosely coupled network of problem solvers can work together to solve problems that are beyond their individual capacities. In MAS, agent behaviors and characteristics are considered. We present some cooperations techniques in the sequence.

2.1.6.1 Task Sharing

The *Task Sharing* is a cooperation technique that consists of a sequence of activities grouped in the following three stages:

- *Problem decomposition*: consists to decompose the overall problem into smaller sub problems recursively until the sub problems to have an appropriate granularity to be solved by an individual agent.
- *Sub problem solution*: the sub problems identified are individually solved. This stage typically involves sharing of information between agents: one agent can help another if it has information that may be useful to the other.
- *Solution synthesis*: solution to individual sub problems are integrated into an overall solution. As in problem decomposition, this stage may be hierarchical, with partial solutions assembled at different levels of abstraction.

The *Task Sharing* takes place when a problem is decomposed in to smaller sub problems and allocated to different agents. The agents share information relevant to their sub problems. This information may be shared pro-actively, that is, one agent sends another agent some information because it believes that the other will be interested in it; or reactively, that means an agent sends another information in response to a request that was previously sent.

2.1.6.2 Result Sharing

The *Result Sharing* is another cooperation technique in which problem solving proceeds by agents cooperatively exchanging information as a solution is developed. It begins from small problems to large ones. Durfee 2001 suggests that problem solvers can improve group performance in result sharing in the following ways:

- *Confidence*: independently derived solutions can be cross-checked, highlighting possible errors, and increasing confidence in the overall solution.
- *Completeness*: agents can share their local views to achieve a better overall global view.
- *Precision*: agents can share results to ensure that the precision of the overall solution is increased.

- *Timeliness*: even if one agent could solve a problem on its own, by sharing a solution, the result could be derived more quickly.

2.1.6.3 Combining Task and Result Sharing

This is an hybrid cooperation technique which consists of dividing the overall problem in to small problems and share knowledge in order to realize a task, therefore, solve the problem. The idea with this hybrid mechanism is to use the benefits of both task sharing and result sharing technique. The task sharing technique reduces the overall problem complexity by decomposing it into small problems. The result sharing technique uses agents knowledge and expertise in order to share solutions as the sub problems resolution performance can be improved.

2.1.7 Coordination

In general coordination can be considered as the way things or activities have to be managed in order to reach some target. Malone and Crowston 1994 present an inter-disciplinary definition of coordination. They consider the coordination as the action of managing dependencies between activities. In the MAS paradigm, Wooldridge 2009 presents the coordination as the mechanism of managing inter dependencies between the activities of agents. For him, in a perfectly coordinated system:

1. Agents will not accidentally clobber each other's sub goals while attempting to achieve the common goal;
2. Agents will not need to explicitly communicate;
3. Agents will be mutually predictable, perhaps by maintaining good internal models of each other;
4. The presence of conflict between agents, in the sense of agents destructively interfering with one other is an indicator of poor coordination.

Numerous coordination mechanisms have been proposed in the MAS and AI literature, exploring several approaches. Schumann 2012 proposed an classification of coordination mechanisms with the idea of presenting an appropriate coordination method, given a MAS scenario or context. This classification is summarized in Table 2.1.

Coordination	Description
Task sharing	Contains approaches that distribute tasks among a number of agents
Auctions	A specific subclass of task sharing mechanism.
Negotiations	Based on structured exchange of messages.
Result Sharing	Enable different experts to collaboratively solve a problem.
GPGP	Generalized Partial Global Planning. A subclass of Result Sharing mechanism
Centralized planning	For the coordination of different agents. This usually lead to very complex planning problems
DPCP	Decentralized Planning for a Centralized Plan. Relies on techniques similar to result sharing
DPDE	Decentralized Planning with Decentralized Execution.
Decoupling	A subclass of DPDE. Tries to add additional constraints to decouple the local problems and allow conflict free combination of of local plans.
Plan merging	A subclass of DPDE. is done by a specific agent that tries to merge locally generated sub plans into a feasible global plan.
Mediators	A subclass of DPDE. try to resolve conflicts between local plans. Similar to merging a specific agent, the mediator, is responsible for resolving conflicts.
Iterative plan formation	A subclass of DPDE. tries to minimize the number of conflicts between sub plans iteratively.
DCSP	A coordination problem is represented as a Distributed Constraints Satisfaction Problem and then solved.
Coordination artifacts	embed a coordination mechanism in the environment in which the agents exist in.

Table 2.1: Group of Coordination Mechanisms, (Schumann 2012).

Wooldridge 2009 also presents some coordination mechanisms introduced briefly in the sequence.

2.1.7.1 Coordination through Partial Global Planning

The main principle of Partial Global Planning (PGP) is that cooperating agents exchange information in order to reach common

conclusions about the problem-solving process. Planning is partial because the system cannot generate a plan for the overall problem. Planning is global because agents form non-local plans by exchanging local plans and cooperating in order to achieve a non-local view of problem solving. PGP involves three iterated stages:

1. Each agent decide what its own goals are and generate short-terms plans in order to achieve them;
2. Agents exchange information to determine where plans and goals interact;
3. Agents alter local plans in order to better coordinate their own activities.

2.1.7.2 Coordination through Joint Intentions

The coordination through joint intention consists of managing cooperative activities between agents through a joint commitment to an overall aim and their individuals commitments to the specific tasks that they have been assigned.

A *commitment* is a pledge or a promise monitored by a means named *convention*. A *convention* specifies under what circumstances a commitment can be abandoned and how an agent should behave both locally and towards others when one of these conditions arises.

As an example of a coordination through join intention, we have a football team.

2.1.7.3 Coordination by Norms and Social Laws

This is probably the coordination mechanism that every human been implicitly use in his everyday lives. As named, this coordination mechanism manages agent activities using norms and social laws. For examples:

- In the traffic, when a driving agent stops by seeing a red firelight;
- In a society, when an agent knock the door before entering into some residence;

Wooldridge 2009 defines a norm as an established expected pattern of behavior. He defines a social law as a norm carrying with it some authority. Convention, as defined in Section 2.1.7.2, is one of the means to establish norms and social laws in MAS. There are two approaches to create a convention in an agent society:

- *Off-line design*: In this approach, norms and social laws are designed before the implementation of the MAS;
- *Emergence from within the system*: Norms and social laws emerge from within a group of agents.

2.2 URBAN TRAFFIC CONTROL (UTC)

In a general context, the urban traffic control (UTC) problem consists in managing the urban traffic system (UTS) in order to supply its demands. Following this context, Burmeister et al. 1997 define the UTC as largely the process of planning, implementation, testing and administration of traffic to optimize the assignment of traffic supplies to traffic demands under economic and environmental features. Similarly, Li et al. 1996 classify the UTC problem in two different views: from *road users*, and *road managers*. From the *road users*, a UTC problem consists only of choosing the direction at the junctions; whereas from the view of *road managers* it is the problem of controlling the traffic flow.

On a micro view of the problem, an UTS is controlled defining, at one side, a traffic network that represents the physical architecture and the dynamic of such UTS. At another side, a control strategy that acts on some components at the traffic network, usually electronic components such as traffic lights, in order to supply a given demand or to reach a desired objective. Generally, a traffic simulator is used in order to model such traffic network and simulate the dynamics of the UTS to be controlled. Depending of the simulator used, the control strategy can be implemented therein or on another system which communicate with the simulator.

2.2.1 Traffic Network

Traffic networks are often modeled using traffic roads, sometimes named *links*, interconnected in *junctions* usually controlled by *traffic lights*. Each link has one or several *lanes* and can be for one direction or bi-directional. To simulate the dynamic of the traffic, vehicles are added in simulations, running on links with some determined speed and acceleration, according to the definition of the scenario being simulated. Depending of the simulator and the simulation scenario, vehicles can be of many types and can travel from an origin to a destination.

Figure 2.6 shows the illustration of a simple traffic network composed of two links and one junction. We use the *Simulation of*

Urban Mobility (SUMO) (Krajzewicz et al. 2012) with the objective to model this traffic network and define the main elements therein in the context of SUMO.

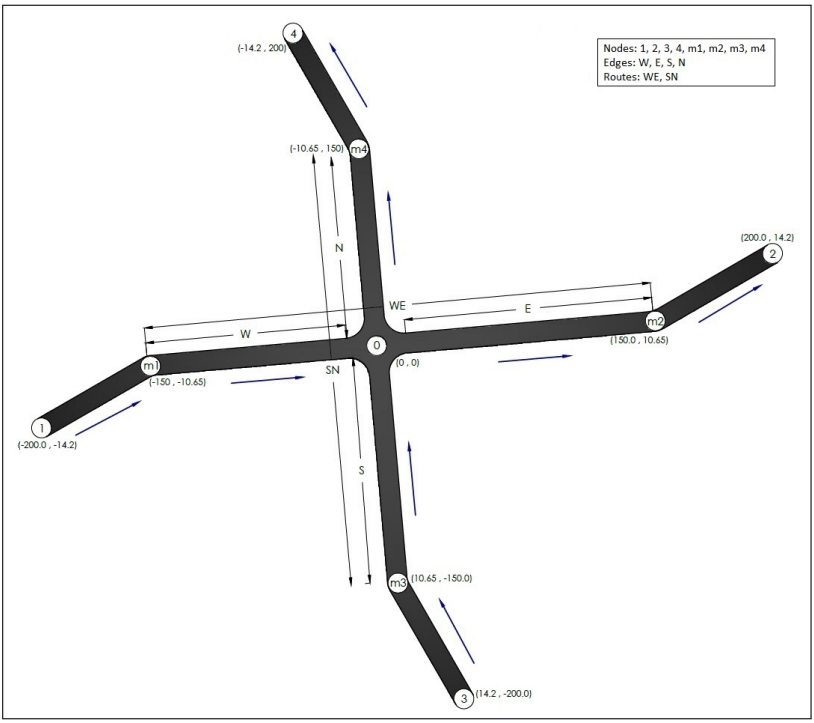

Figure 2.6: Example of a traffic network.

(Krajzewicz et al. 2012) presents SUMO as a free and open traffic simulation created by the Institute of Research in Transportation of the Germany Aerospace Center, available since 2001. SUMO allows modeling of inter-modal traffic systems including road vehicles, public transport and pedestrians. A traffic simulation using SUMO occurs in the interval of time which the unit is a *step*, equivalent to one second by default.

We present in the sequence the main elements that compose an urban traffic system, illustrating these elements in the SUMO context, as well as defining some main attributes of each element. (more details can be found in the SUMO documentation [1].

[1] <http://sumo.dlr.de/wiki/Main_Page>

2.2.1.1 Traffic Junction

A *junction* is an intersection between two or several links. A junction can be controlled by a set of traffic lights, by others traffic signs such as horizontal and vertical signs, or only by traffic laws that vehicles have to follow.

In SUMO, a junction is modeled defining a *node* $N = (x, y)$, which is a bi-dimensional point into the network. Each node has the following attributes:

- id: identifies the node into the network.
- x: represents the x-coordinate value of the node.
- y: represents the y-coordinate value of the node.
- $type$: is an enumeration of node type *(priority, traffic light)*.

The *priority* type makes that vehicles on links with less priority wait first at the junction represented by the respective node. The *traffic light* type makes the junction represented by the respective node to be controlled by a traffic light.

2.2.1.2 Traffic Link

A *link* is a segment of a road, between two points. These points can represent the start or/and the end of this road, or intersections with links of others roads. A link can have one or several *lanes*.

In SUMO, Traffic links are created connecting nodes and specifying the number of lanes therein, as well as others elements that describe the dynamic of the traffic in the link. In SUMO, a traffic link has the following main attributes:

- id: identifies the link into the network.
- $from$: represents the id of the source node of the link.
- to: represents the id of the target node of the link.
- $numlanes$: specifies the number of lane in the link.
- $speed$: specifies the maximum speed that vehicles can reach on the link.
- $priority$: specifies the link priority in relation to others link in the network.

With the definition of the nodes and links, the physical architecture of the traffic network can be represented. The dynamic of the traffic system is defined with the representation of routes with its respective traffic flows as well as by vehicles with their respective trip which can be random or specified. We describe the traffic routes and vehicles in the sequence.

2.2.1.3 Traffic Light

Traffic lights are electronic components that control the crossing of vehicles at junctions. One junction can be controlled by two or a group of traffic lights working in coordination.

In SUMO, traffic lights can be defined automatically with the generation of the traffic network based on the specification of nodes and links. Traffic lights have the following main attributes:

- id: identifies the traffic light into the network.
- $phase$: represents the traffic light cycle. A phase comprises:
  - $duration$: represents the duration of the phase.
  - $state$: represents the traffic light states for this phase.
  - $minDur$: represents the minimum duration of the phase when using type *actuated*.
  - $maxDur$: represents the maximum duration of the phase when using type *actuated*.
- $programID$: identifies the traffic light program.
- $type$: is an enumeration type of the traffic light.
  - $fixed$: set the fixed cycle durations.
  - $actuated$: set the cycle prolongation based time gaps between vehicles.
- $offset$: represents the initial time offset of the program.

The phase states are represented by a list of characters (letters) that determine the signs at traffic lights. We describe in the sequence the main characters used in this dissertation.

- r: *red light* for a signal - vehicles must stop.
- y: *yellow light* for a signal - vehicles will start to decelerate if far away from the junction, otherwise they pass.

- g: *green light* for a signal, no priority - vehicles may pass the junction if no vehicle uses a higher prioritized link, otherwise they decelerate for letting it pass. G: *green light* for a signal, priority - vehicles may pass the junction.

Traffic lights can be also added manually by defining a *additional* file.

2.2.1.4 Traffic Route

In SUMO, traffic routes are created by connecting links according to the desired trajectory to be traveled by vehicles. To define a route, we have to specify the sequence of links that compose such route and the traffic flow on the route. A traffic route has the following main attributes:

- id: identifies the route in the simulation.
- $edges$: specifies the sequence of links that compose the route.
- $flow$: represents the traffic flow on the route, with the following attributes:
  - id: identifies the flow in the traffic system.
  - $route$: represents the id of the route with the respective flow.
  - $begin$: specifies the initial number of vehicles on the route.
  - end: determines the total number of vehicles that will travel on the route.
  - $period$: determines the time interval of the travel start of vehicles on the route.

2.2.1.5 Traffic Vehicle

There are several ways to define vehicles in SUMO. We present here the definition used in the implementation of our simulation. SUMO allows the definition of vehicle types through the *vTypeDistribution*, with the following attributes:

- id: is an enumeration that specifies the type of vehicle distribution.
- $vType$: describe the vehicle characteristics, with the following attributes:

- id: is an enumeration that specifies the type of vehicle.
- $length$: specifies the vehicle length in meters.
- $accel$: specifies the maximum acceleration of the vehicle in (m^2/s).
- $deccel$: specifies the maximum deceleration of the vehicle in (m^2/s).
- $maxspeed$: determines the maximum speed of the vehicle in (m/s).
- $sigma$: determines the driver imperfection. $sigma \in [0, 1]$.
- $probability$: determines the running probability of the vehicle type on the route.

2.2.2 Traffic Control Strategy

Usually, UTC strategies are proposed in order to reach a specific objective such as: minimize the congestion: (Min and Jin 2013), (Stefanello et al. 2013), (Kosmatopoulos et al. 2007), and (Kartalopoulos 1999); optimize the collective transportation: (Zeng et al. 2014) and (Jeng et al. 2013); improve some specific team dislocation: (Sundar et al. 2015) and (Viriyasitavat and Tonguz 2012); etc. Most of these strategies are based on the dynamic of traffic lights and tend to modify or even simplify such dynamic with the objective of optimizing some variables that are part of the traffic dynamic such as the traffic flow, density, travel time, main speed, etc.

We introduce first some concepts that are part and characterize the UTS and UTC, focusing on the traffic lights control, following the Brazilian Signalization Guideline (DENATRAN 2014).

2.2.2.1 Traffic Movements

A traffic *movement* at a junction is used to identify vehicle flows with the same origin and destination. There are *vehicular* and *pedestrian* movements. Figure 2.7 shows the example of both of the movements.

Two movements $M1$ and $M2$ are considered *converging movements* when they are from different origins and have the same destination. $M1$ and $M2$ are considered *diverging movements* when they are from the same origin and have different destinations. $M1$ and $M2$ are *intercepting movements* when they are from different origins and destinations, and they cross at a given point of the junction.

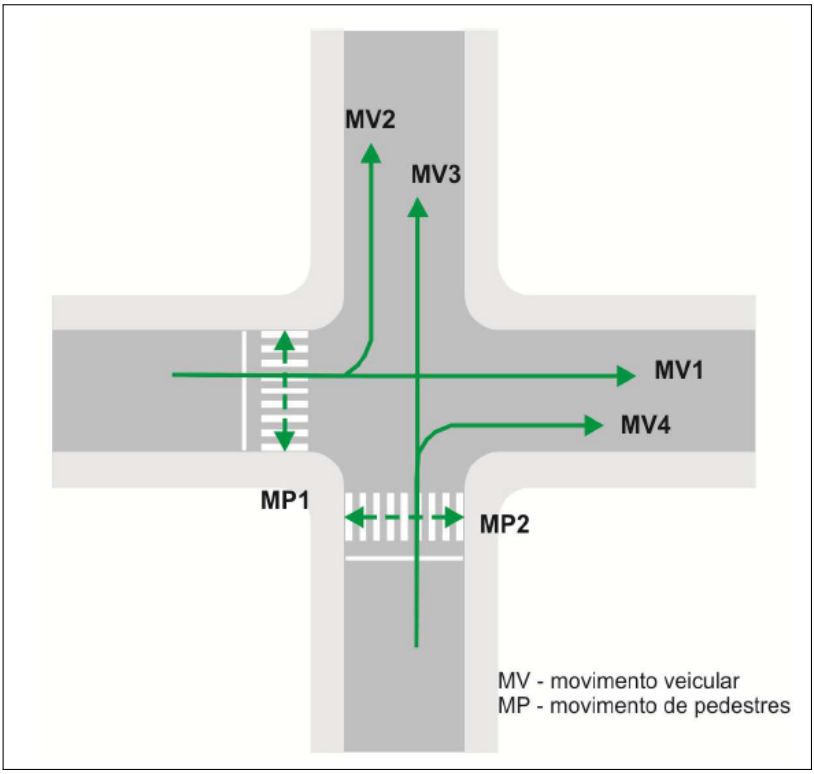

Figure 2.7: Example of movements at a junction.

(DENATRAN 2014)

Lastly $M1$ and $M2$ are *non-intercepting movements* when they are from different origins and destinations, and they do not cross at any point of the junction. Figures 2.8 show the respective movements.

Converging and *intercepting* movements are considered *conflicting movements*. whereas, *diverging* and *non-intercepting* movements are considered *non-conflicting* movements.

A *diagram of conflicts* consists on the schematic representation of a junction geometry, considering all the vehicular movements that can occur therein. Figure 2.12 shows an example of a diagram of conflicts at a junction with 4 links and 16 possible vehicular movements.

A *table of conflicts* highlights conflicting movements from the diagram of conflicts. Figure 2.13 shows the table of conflicts result-

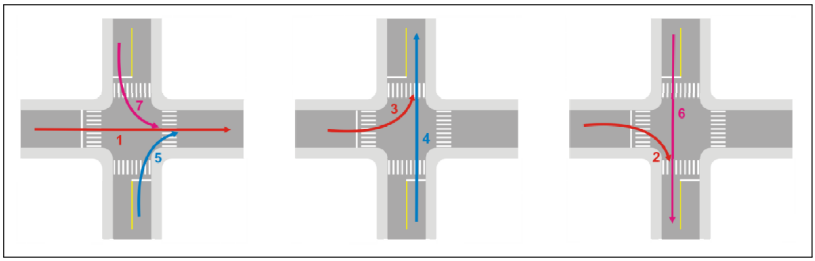

Figure 2.8: Example of converging movements.

(DENATRAN 2014)

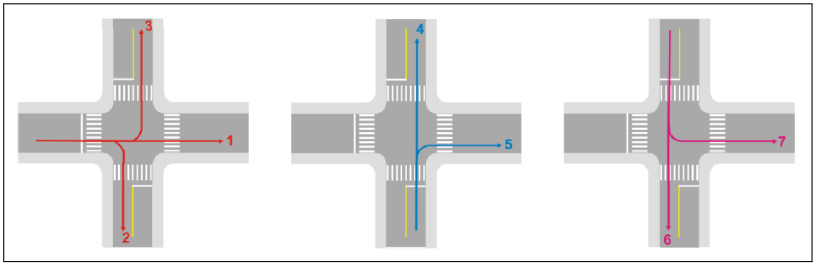

Figure 2.9: Example of diverging movements.

(DENATRAN 2014)

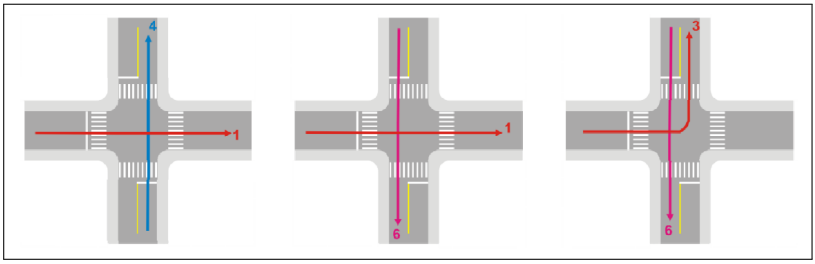

Figure 2.10: Example of intercepting movements.

(DENATRAN 2014)

ing from the diagram shown in Figure 2.12.

2.2.2.2 Some Control Concepts

A *group of movements* is a set of movements from a same link that receive simultaneously the right to cross the junction.

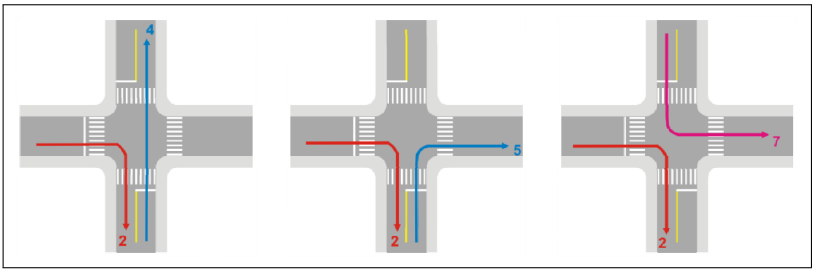

Figure 2.11: Example of non-intercepting movements.

(DENATRAN 2014)

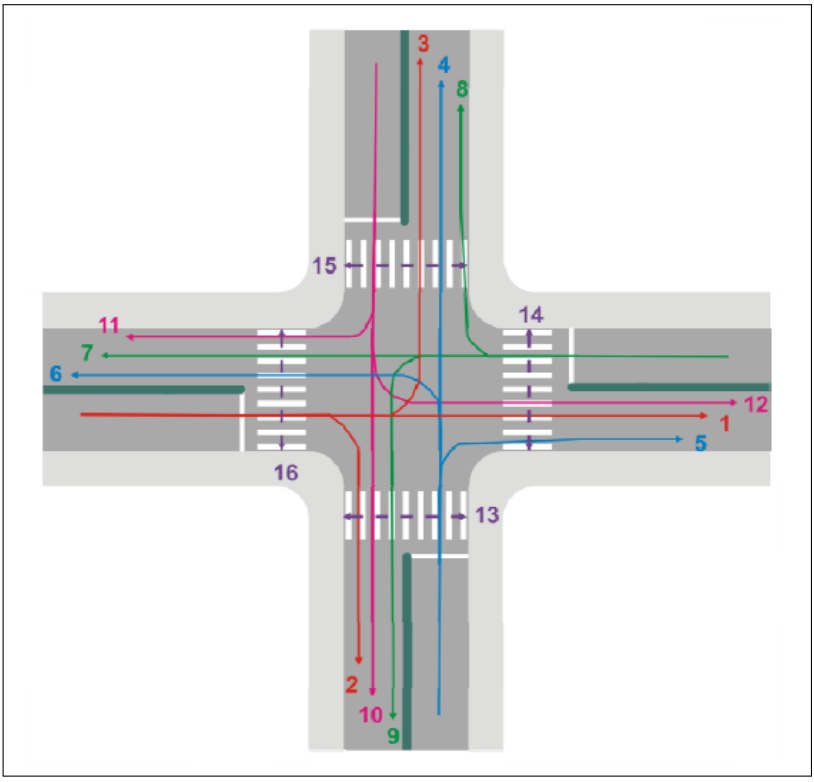

Figure 2.12: Example of a diagram of conflicts.

(DENATRAN 2014)

MOV.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1				x	x	x			x	x		x		x		x
2									x	x			x			x
3				x		x	x	x	x	x		x			x	x
4	x		x				x	x	x			x	x		x	
5	x											x	x	x		
6	x		x				x		x	x	x	x	x			x
7			x	x		x				x	x	x		x		x
8			x	x										x	x	
9	x	x	x	x		x				x		x	x	x		
10	x	x	x			x	x		x				x		x	
11						x	x								x	x
12	x		x	x	x	x	x		x					x	x	
13		x		x	x	x			x	x						
14	x				x		x	x	x			x				
15			x	x				x		x	x	x				
16	x	x	x			x	x				x					

Figure 2.13: Example of a table of conflicts.

(DENATRAN 2014)

A *group of signs* is a set of traffic lights with the same sign that control a given group of movements.

A *Stage* is the interval of time in which one or several groups of movements receive simultaneously the right to cross the junction. A stage comprise the time of the green sign and the time interval until the next green sign occurs. Figure 2.14 shows the example of two consecutive stages.

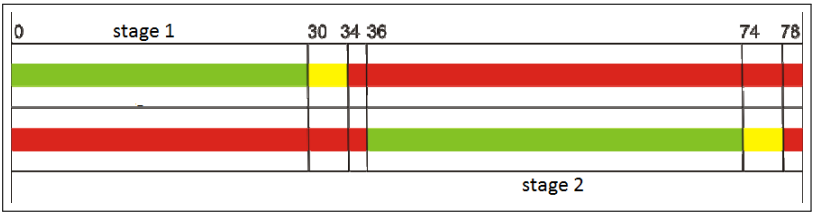

Figure 2.14: Example of two stages.

A *cycle* is the sequence of all the stages of a traffic light controlling a junction. The time of a cycle corresponds to the sum of the time of all the stages therein.

A *interval* is a period of time in which the configuration of traffic signs that control a given link stay unaltered.

A *traffic signal plan* is a set of elements that characterize the traffic lights signalization programmed at a junction during a time period.

A *diagram of stages* is a graphic representation of all the movements that can be realized in each stage during a cycle. Figure 2.15 shows the example of a diagram of stages.

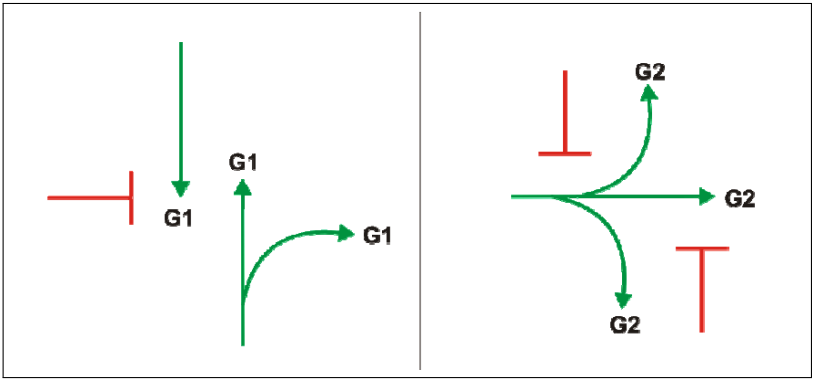

Figure 2.15: Example of a digram of two stages.

(DENATRAN 2014)

2.2.2.3 Control Types

There are two types of traffic light control systems regarding the traffic signal plan: *Fixed-time* control and *Actuated* control. *Fixed-time* control uses traffic signal plans calculated according to the historical data of some traffic variables such as the traffic flow, average speed. During *fixed-time* control the cycle, sequence of stages, and duration of signs are constant. *Fixed-time* control can be realized based on one traffic signal plan, or based on several plans according to the traffic conditions of different periods of the day.

Actuated control is classified in *semi-actuated* and *totally actuated* control. The *semi-actuated* control is often used at junctions between links with high traffic volume (main links) and links with low traffic volume (secondary links). The traffic signal plans at these junction are calculated in order to prioritize the mains links until the detectors at the secondary links notify the presence of vehicles at the secondary links.

The *totally-actuated* controls is based on the determination of the green time of each stage during a cycle of a traffic signal plan.

The green time is determined dynamically and at real time, between an interval of a minimum and maximum values established, according to the traffic demand and conditions. The *totally-actuated* control allows the set of some parameters such as the link priority, in real time, with the objective of suppling a given demand like the presence of emergency vehicles.

2.2.2.4 Control Strategies

There are two main types of control strategies with respect to the traffic network: the *Isolated* control and the *Network* control, also called *coordinated* control. In *Isolated* control, each junction is controlled independently of the others. There is no coordination among traffic lights at junctions. In this case, the programming of traffic lights considers only the traffic state at the respective junctions. *Isolated* control can affect seriously the traffic flow performance in situations where junctions are close to each other.

Network control allows the programming of traffic lights not only at each junction, but also considering the traffic dynamic at the global network. The objective is to ensure a good traffic flow performance both at junctions individually and the global performance of the traffic network.

2.2.2.5 Control Modes

There are two main modes to control traffic lights. The *decentralized* control and the *centralized* control. In the *decentralized* control mode, the programming of traffic lights is implemented into the controller at the traffic light. The *centralized* control mode is implemented in a central computer or group of computers.

2.3 EMERGENCY MANAGEMENT

Emergency management is a process that begins when some emergency center receives the notification of an incident classified as an emergency, and ends when such situation is recovered. Borges et al. 2010 consider typically three stages involving an emergency management:

1. *Notification and validation*: starts when an incident notification is reported. In this stage, the call center usually identifies the emergency situation in order to verify if the crisis is real.

2. *Response process*: takes place once the crisis is verified as true. It involves planning, triggering, monitoring and adjusting response activities which consist in dispatching emergency vehicles, assigning equipment and personnel, etc.
3. *Closing activity*: involves basically reporting the result of the response process.

Emergency management usually involves several teams working together. Some of the teams generally present are the police, the fire-fighters, and the medical personnel. Each team often has its particular objectives that compound the overall goal. Generally, the police attempts to control and ensure the order and security of the incident location. The fire-fighters usually attempt to save lives and protect property, (Trent et al. 2008). The medical personnel attempts to save a maximum of lives and reduce the maximum number of human and property injuries.

Each emergency team usually follows its own action protocol, nevertheless they always try to combine these protocols in the benefit of the established overall goal. Trent et al. 2008 presents five functions found in fire-fighters response process.

1. *Manage Routes*: includes planning and executing movement to and from the incident and includes negotiating local paths at the incident.
2. *Manage Resources*: includes monitoring, committing, requesting and withdrawing personnel, equipment and supplies.
3. *Reduce Threats*: entails extinguishing, containing or dissipating fire, hazardous material or other environmental hazards to life and property.
4. *Situation Assessment*: includes gathering intelligence, monitoring, and assessing the state of the threat, and progress or effectiveness of the response.
5. *Extraction*: includes removing incapacitated fire-fighters from danger.

Moreover, they entail to each function, the decisions that support it, as well as the information required for this decisions. We summarize these decisions and information requirements in Table 2.2 for the function *manage routes*, in Table 2.3 for the function *manage resources*, in Table 2.4 for the function *reduce threats*, in Table 2.5 for

Decisions	**Requirements**
-What route to take for approaching the incident? -Where to lay hose lines? -What are the valid entry/exit paths? -Does a path need to be created?	-Infrastructure limitations -Traffic patterns -Routes of other responders -Environmental conditions -Occupancy status Confirmed life hazard -Condition of roof -Locations of: - -Incident - -Water sources - -Fire or contamination - -Extensions of fire or contamination - -Elevators, stairs, doorways, access points - -Obstacles for entry

Table 2.2: Fire-fighter Function: Manage Routes, (Trent et al. 2008).

the function *situation assessment*, and in Table 2.6 for the function *extraction*.

Such as fire-fighters, the police and medical personnel also have some strategies to cope with emergency response processes.

Decisions	Requirements
-When and where to commit resources? -When to withdraw or replace resources? -When to request resources? -Who to designate as a safety team? -Where to establish command post and staging areas? -When to request casualty coordinator? -How to position ladders and pumps?	-Progress of search -Conditions in building -Occupancy status -Water supply -Resource depletion -Expertise/Trust in working groups -Time units have been exposed -Current staffing levels -Unique apparatus available -Status of uncommitted units -Emergency responder casualties -Structure type and floor plan -Street conditions -Locations of: - -Fire or contamination - -Extensions of fire or contamination - -Life hazards - -Resources - -Power lines - -Water sources - -Building entrances - -Other vehicles

Table 2.3: Fire-fighter Functions: Manage Resources, (Trent et al. 2008).

2.4 SOME WORKS IN THE LITERATURE

The topic studied in this dissertation, *Traffic Control Under An Emergency Response*, involves at least two mature topics in the literature which are: *Traffic Control* (McCluskey et al. 2016); (Guberinic et al. 2007); (Riedel and Brunner 1994) and *Emergency Response* (Fagel 2010); (Dillon 2014). Although numerous researches have been proposed in both fields, this dissertation analyses specifically solutions that use a MAS approach. This is the main topic of interest, since in the next chapter, we propose a MAS model that implements a traffic control strategy and manages and routes emergency vehicles in order to improve an emergency response process. We briefly comment some works that propose solutions from different areas than MAS.

Google Scholar, IEEE Explorer, and Mendeley were used as biography reference managers and academic social networks in order to find works related to the studied topic, combining key words such as: *Traffic Control*, *Emergency Response* and *Multi Agent Sys-*

Decisions	Requirements
-Whether to attack or contain the fire? -Reduce or contain contaminants? -Need to set up/establish decontamination? -Whether to ventilate or not? -What substance(s) to use on contaminants or fire? -Where to attack threat?	-Structure type and floor plan -Conditions in building -Type of contamination -Surrounding population -Weather effects on contaminants -Locations of: - -Fuel sources - -Fire or contamination - -Extensions of fire or contamination - -Hose lines - -Scuttles and skylights

Table 2.4: Fire-fighter Function: Reduce Threat, (Trent et al. 2008).

Decisions	Requirements
-Is it a false alarm? -Cease or continue search for life? -Cease of continue search for fire? -Where to search?	-Source of alarm -Reports from occupants -Presence of heat or smoke -Fire containment -Occupancy status -Progress of search -Exposures -Structure type and floor plan -Potential for flash over/ back draft -Resource depletion -Time of day -Locations of: - -Fuel sources - -Fire or contamination - -Extensions of fire or contamination - -Hose lines - -Scuttles and skylights - -Stairs - -Life hazard - -Small rooms

Table 2.5: Fire-fighter Functions: Situation Assessment, (Trent et al. 2008).

tems. We summarized some results in Table 2.7 in Section 2.4.1, relative to works that use MAS approach, and Table 2.8 in Section 2.4.2, for other approaches. Both Table 2.7 and 2.8 comprise a *title*, *specific scenario* and *field* columns. The field refers to the domain from where the solution method used in the publication derive.

Decisions	Requirements
-Focus on threat reduction or rescue? -Where to establish a safe refuge area? -What is the best method for evacuation? -When to deploy a rescue team?	-Presence of ladder company -Water supply -Conditions in building -Occupancy status -Location of: - -Fire or contaminants - -Extension of fire or contaminants - -Stairs, balconies, fire escapes, elevators, exits - -Rescue teams - -Incapacitated or Lost emergency responder

Table 2.6: Fire-fighter Functions: Extraction, (Trent et al. 2008).

2.4.1 Works Using MAS Approach

Table 2.7 summarizes works that propose solutions relatives to the traffic control and emergency response problem using a MAS model. Some of these works use MAS only for simulation purposes, applied to the emergency response scenario. Some of these works are described with more details in the sequence.

2.4.1.1 Agent Based Simulation Applied to the Design of Control Systems for Emergency Vehicles Access

The (Patrascu et al. 2015) work aimed to formulate an agent based simulation perspective on aiding the design of complex and distributed control systems, such as the one facilitating the arrival of the emergency vehicles to accident sites. In this way, they introduced the *tissue*, a type of multi-agent used to construct *Agent Based Simulation Models* (ABSM)s in order to test and validate complex control systems, in particular, traffic control and emergency vehicle access in urban areas. A conceptual representation of the *tissue* MAS is depicted in figure 2.16.

They presented some concepts used in their paper as follow:

- A *tissue* is a set of cells agent of the same abstraction level, with the same goal, and concurrent objectives.
- The *tissue* goal is to maintain access for emergency vehicles with minimal disturbance to regulate traffic.

Title	Scenario	Field
Agent Based Simulation Applied to the Design of Control Systems for Emergency Vehicles Access (Patrascu et al. 2015)	Agent based simulation perspective for the design of complex and distributed control systems	Multi Agent System
A multi-agent system based approach to emergency management (Mala 2012)	Decision making support in Emergency Managements	Multi Agent System
Hardware-In-the-Loop Simulation of DC Microgrid with Multi-Agent System for Emergency Demand Response (Yoo et al. 2012)	Simulation of DC Microgrid with MAS for Emergency Response Demand	Power Energy, Hardware In the Loop (HIL), Multi Agent System
A Multi Agent System for Traffic Control for Emergency by Quadrants (Rodríguez et al. 2009)	Traffic control under emergency situations	Multi Agent System
A Cooperative Multi Agent System For Traffic Management And Control (Tomás and Garcia 2005)	Traffic control under emergency situations	Multi Agent System

Table 2.7: Works that use MAS approach.

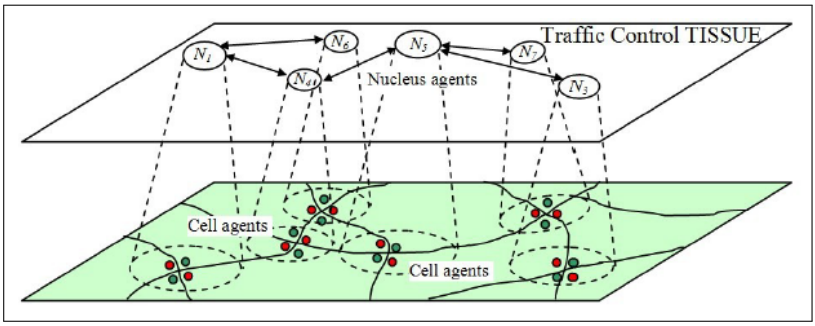

Figure 2.16: Tissue MAS concept (Patrascu et al. 2015).

- A *minimal cell agent* comprises a set of inference devices D_I, sensorial devices D_S, and actuating devices D_A, organized around a nuclear agent N.

- A *nuclear agent* is composed of a communication unit and a (set of) algorithm(s) for cell aggregation/desintegration which serve to form cell agents dynamically within the system.

The (Patrascu et al. 2015) also considered the traffic control, defining the traffic control problem as given an urban area, composed of a network of roads, it is required that the traffic in the area is fluid for regular participants and that emergency vehicles have the fastest possible access through said traffic on their designated routes.

They described the mains components of the control system, being:

- *Plant*: the network of intersections (actual inner crossing area and the adjacent road sections directed inward) with moving vehicles;
- *Plant outputs*: number of cars on the adjacent road sections waiting to pass through the intersections;
- *Plan inputs*: traffic lights signals;
- *Main requirement*: minimization of emergency vehicles waiting times;
- *Secondary requirement*: minimization of all vehicles waiting times, with regards to giving priority to emergency vehicles.

They considered the controlled system as a *dynamically changing system fo routes*, in which each route has the traffic signals as inputs and the vehicle waiting times as outputs. For each intersection P_i, within a route of a vehicle, the control system can be designed as depicted in figure 2.17, where C_i is the control algorithm, y_i is the vehicle waiting time, u_i is the combination of traffic lights signals relevant to the direction the vehicle intends to travel through. The desired outcome (setpoint r_i of the system) is zero waiting time (signal ϵ_i, is the control derivation).

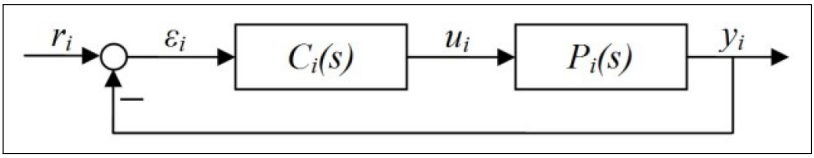

Figure 2.17: Vehicle in intersection control loop structure (Patrascu et al. 2015).

They designed one minimal cell agent ffiICCffl$i$ for each intersection, depicted in figure 2.18, in which: $D_I i$ is an inference

device running the control algorithm; D_Ai are acting devices (the traffic lights ensemble); D_Si are sensing devices (smart sensors mounted along the adjacent road sections that detect the incoming vehicles and their types); N_i cell nucleus; nv_i number of vehicles waiting to pass through (raw environmental data); y_i processed sensor data (including vehicle type); u_i computed command vector containing the traffic lights color combination; tl_i displayed traffic light colors (executed commands); obj_i cell objective of controlling the intersection received by nuclei a priori; tsk_i desired tasks according to objectives: the D_S task is to collect and process data, then transmit it to the D_I; whose task is to elaborate control decisions for the D_A; who in turn are tasked with executing the commands using their visual actuators.

All intersection control cells ffiICCffl$i$ are part of the traffic control tissue ´TCTˇ. The *tissue* goal is to maintain fluid traffic throughout the road network with emergency vehicle priority, whereas the *cell* objective is to minimize delays within their assigned intersection. The control system is completely distributed: each D_I elaborates a control decision individually.

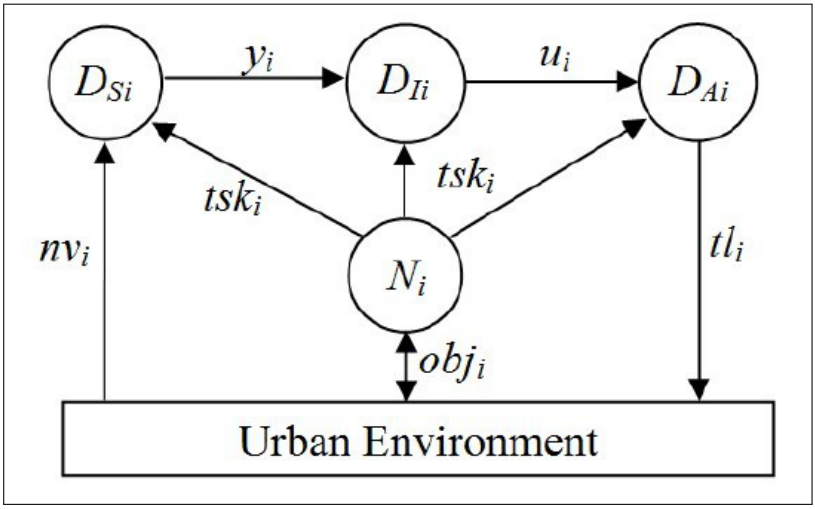

Figure 2.18: Intersection control cell structure (Patrascu et al. 2015).

Comparing to our dissertation contribution, (Patrascu et al. 2015) approaches the emergency response process using MAS and

traffic control concepts as done in our dissertation. Moreover the objectives of the systems established in their work are almost the same considering our objective. The only difference is that their secondary objective was to minimize the normal vehicles waiting time whereas ours was to to reduce the impact of priorities given to responders. These to objectives can seem to be similar, however, the difference remains in the fact that our dissertation does not improve the traffic flow for normal vehicles.

However, (Patrascu et al. 2015) does not approaches the problem of best path for responders displacement. We consider fundamental to provide a best path for responder displacement, since we need to improve their travel time through the traffic. In (Patrascu et al. 2015) solution, such problem is not clearly approached.

2.4.1.2 A Multi-Agent System Based Approach to Emergency Management

(Mala 2012) proposed a complex adaptive system approach to emergency management decision support systems via an agent based application. They presented the emergency management problem in a macro concept depicted in figure 2.19.

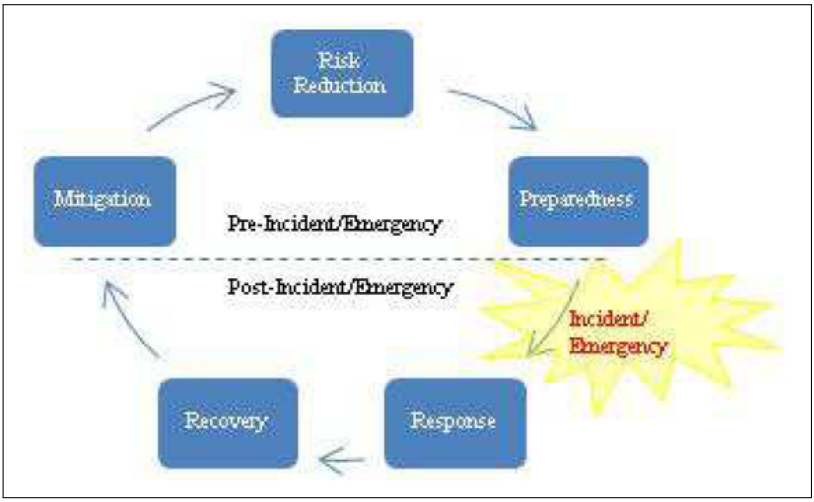

Figure 2.19: Emergency Management Macro Concept (Mala 2012).

They proposed a reactive agent based multi-agent system approach to manage activities focusing on the response and recovery

process of an emergency management. As depicted in figure 2.20, the whole system is represented in seventeen reactive agents with limited planning capacity and knowledge about their environment. These agents are designed with simple roles to react to environmental and inter-agent input.

Lastly, a data warehouse integrated with databases and geographic information system provide information to the whole system. These databases contain information such as location, strength availability, readiness, time, etc. Data in these databases are updated near real time from external resources.

However, this work did not explained deeply the characteristics and objectives of each agent, or how agents interact each one to another. Any interaction protocol was presented in order to show how the interactions between the agents would occurs. They mentioned the environment without presenting its characteristics, as well as, how the agents interact with such environment. All of these observations are approached in our dissertation.

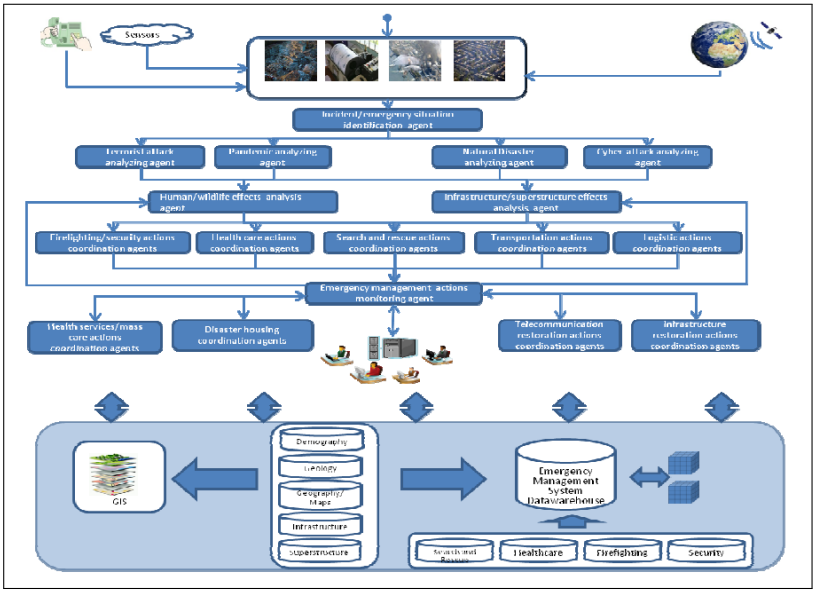

Figure 2.20: Proposed Multi-Agent System (Mala 2012).

2.4.1.3 A MAS for Traffic Control for Emergencies by Quadrants

Rodríguez et al. 2009 proposed a MAS model with the objective of allowing emergency vehicles to get through reasonable routes from one origin point to some desired destination, safely and on time. In their proposal, they divide an urban city map into quadrants of similar size that represent small sections of the respective city, considering that no traffic zone is excluded.

Rodríguez et al. 2009 classify three types of agents considering the area in which the agents act and the operations to be controlled by these agents.

- *Individual Agents*: represent emergency vehicles. They communicate with other agents to follow the instructions provided by them.
- *Local agents*: represent cameras and traffic signs. Cameras act as environment control agents. They display one or more areas or traffic status in real time. Traffic signs can be opened or closed as desired;
- *Regional Agents*: responsible for receiving requests from the various emergency services. Such services indicate the data required for the emergency vehicle movement and interact against the agents in order to establish the "action plan" or best way to got to the emergency location.
- *Global Agents*: are in the higher level of command in terms of the system architecture. They are responsible for managing the rest of the agents of each area.

There are also other traffic electronic components which are not agents, but are used as objects with some role in the system. Most of these components are used to get or display information. These components are described as follow.

- *Information panels*: inform the people of actions needed, such as opening a traffic light, or clearing a path for an emergency vehicle that requires its use;
- *Sensors*: detect surrounding objects which must be considered, such as those which would turn on or off lights, billboards or any kind of obstacle in order to remotely let the authorized vehicle approach the emergency area;

This work does not consider the environment level explicitly. However, in our conceptions such elements could represent artifacts being part of the environment in which the agents cited above act. Figure 2.21 shows the map division by quadrants with its respective agents.

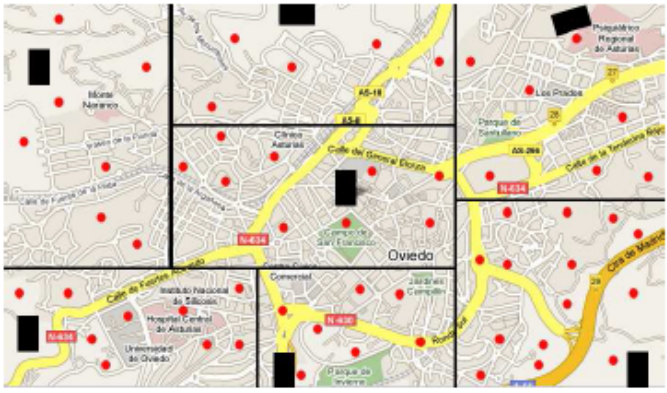

Figure 2.21: Example of a MAS Architecture Prototype (Rodríguez et al. 2009).

Lastly, Rodríguez et al. 2009 define a system topology as well as an interaction protocol as a language for agents communication. Figure 2.22 shows such prototype and interaction protocol.

2.4.1.4 A Cooperative MAS for Traffic Management and Control

Tomás and Garcia 2005 present a MAS model for traffic management and control under a meteorological incident. They exposed the main features of such MAS (its ontology, agents and interaction protocols) and the behavior exhibited by such MAS when a meteorological incident is detected by a traffic administration in a real traffic scenario, case of the A3 Spanish free-way. They begin defining a road traffic domain ontology composed of three sub domains:

- *The road sub domain*: defined by roads, itineraries, segments, and links, each one explained in detail in their work;
- *The behavior sub domain*: describe the traffic behavior as well as its related parameters, and is classified into two main groups: (1) traffic parameters, and (2) weather parameters;

PROTOCOL MESSAGE	EXPLANATION
SU→AG1[ROUTE FROM "A" TO "B"]	Urgent service communicates with AG1. Route from "A" to "B" is needed.
AG1→SU[ROUTE: P1,P3,P2,P5]	AG1 answers. Route goes through P1,P3,P2,P5.
AG1→AG7[WARNING:STAY LEVEL 1]	AG1 warns AG7. Alert level 1.
AG1→AG3[WARNING: STAY LEVEL 2]	AG1 warns AG3. Alert level 2.
AG1→AG2[WARNING: STAY LEVEL 2]	AG1 warns AG2. Alert level 2.

Figure 2.22: Example of Protocol Messages, (Rodríguez et al. 2009).

- *The equipment sub domain*: composed by data capture station, which is defined by traffic data capture station and meteorological station, CCTV cameras, emergency phones, and signals;

Figures 2.23, 2.24, and 2.25 show the three ontology sub domains respectively.

They define a MAS architecture composed of several kinds of agents: Meteo agents, Manager agent, XML Plan agent, Web agent, DF agent and Interface agent.

- *Meteo Agent*: monitor the weather evolution over time in concrete road network areas through specific sensors;

- *Manager Agent*: provide support to the road traffic operator. This support is focused on the determination of the TMP scenarios and on the definition of the dynamic measures to be developed;

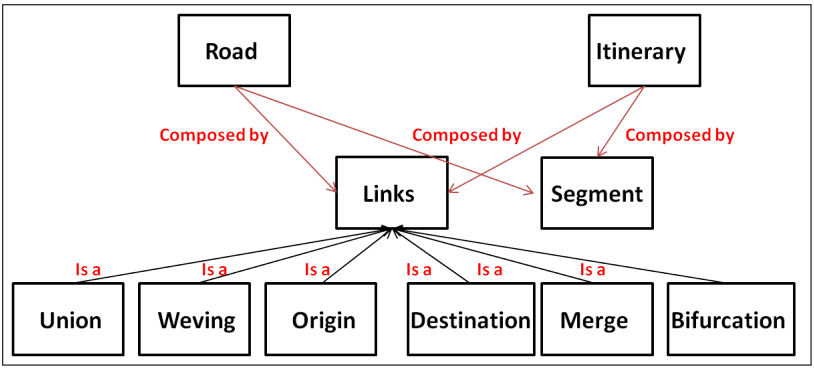

Figure 2.23: Road Sub Domain Ontology, (Tomás and Garcia 2005).

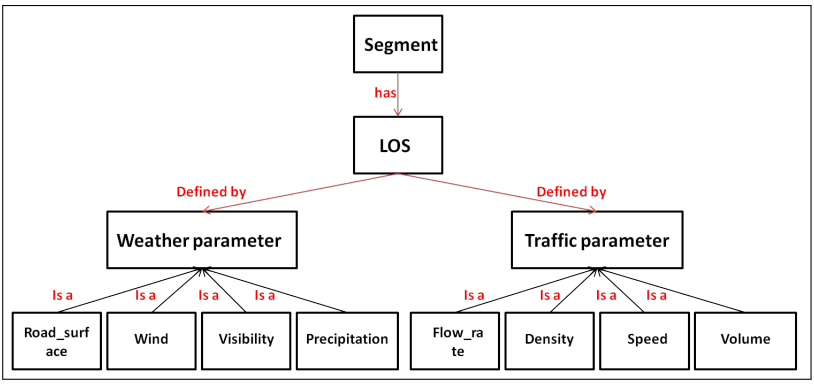

Figure 2.24: Behaviour Sub Domain Ontology, (Tomás and Garcia 2005).

- *Interface Agent*: communicate the road manager with the MAS and display graphically all the components of the MAS: road network, equipment status, TMPs, etc;
- *XML Plan Agent*: contains a database with the TMPs in XML format. When this agent receives information about a detected incident, it looks for the associated fired event in the database. If a TMP for this incident exits, the agent returns to the manager agent the traffic measures to be applied;
- *Web Agent*: the web agent translates the incident information received from the manager agent in a DATEX format (a stan-

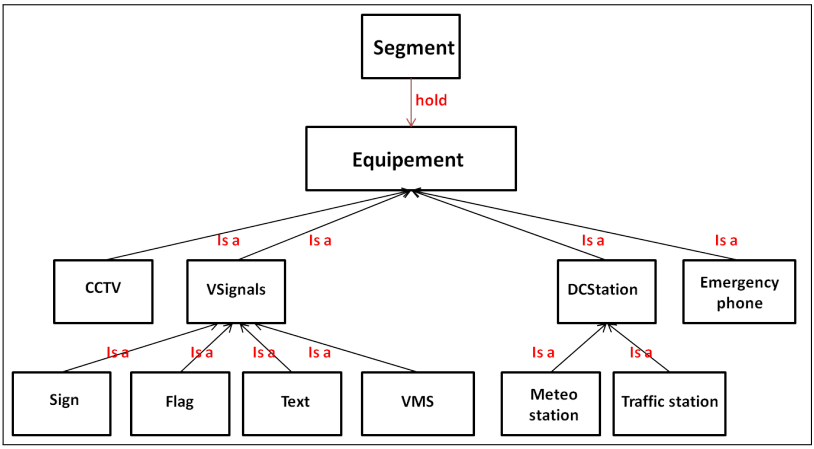

Figure 2.25: Equipment Sub Domain Ontology, (Tomás and Garcia 2005).

dard format used to exchange traffic information between traffic control centers);

- *DF Agent*: the directory facilitator used is the DF JADE agent specified by FIPA. The DF provides a yellow pages service by means of which an agent can register, de register and search for other agents or services in the MAS environment.

Figure 2.26 shows the software architecture of this MAS prototype.

Finally, they define the interaction protocol as the way agents can communicate between them. Such protocol follows the FIPA-ACL interaction protocol defined by FIPA. The communication protocols are classified in two groups: Inner protocols used in communications inside the MAS prototype and Outer protocols used in communications between the MAS and the road traffic operator via the interface agent.

The inner protocols are:

- *ManagerReg*: subscription protocol between the Manager agent and the DF agent;
- *MeteoReg*: request protocol between the meteo and DF agents;
- *AlarmReg*: subscription protocol between the manager agent and a meteo agent;

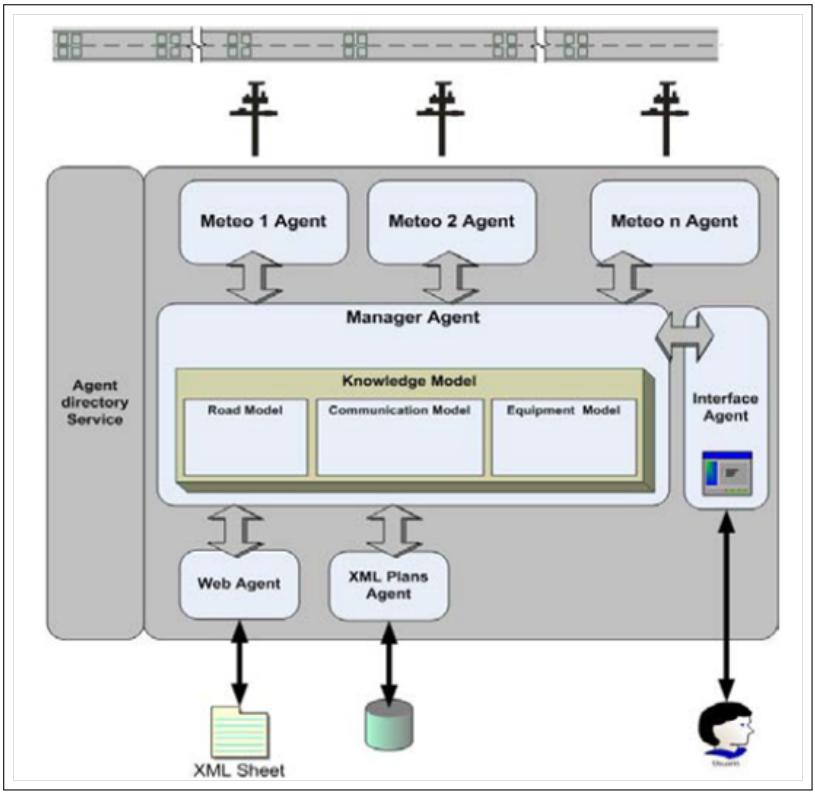

Figure 2.26: Example of a MAS Architecture Prototype, (Tomás and Garcia 2005).

- *PlanMesures*: request protocol between the manager agent and the XML Plan agent;

- *WebInfo*: request protocol between the manager and the web agents;

The outer protocols are:

- *ShowPlan*: request protocol between the manager agent and the interface agent to show the measures of a TMP to be activated;

- *ShowSignals*: request protocol between the manager agent and the interface agent to show the message to be put in a variable signal;
- *ForceSignal*: request protocol between the interface agent and the manager agent to force a message to be put in a variable signal;
- *ValidateScenario*: query protocol to validate a scenario proposal.

Figure 2.27 shows the example of ManagerReg and MetreoReg protocols.

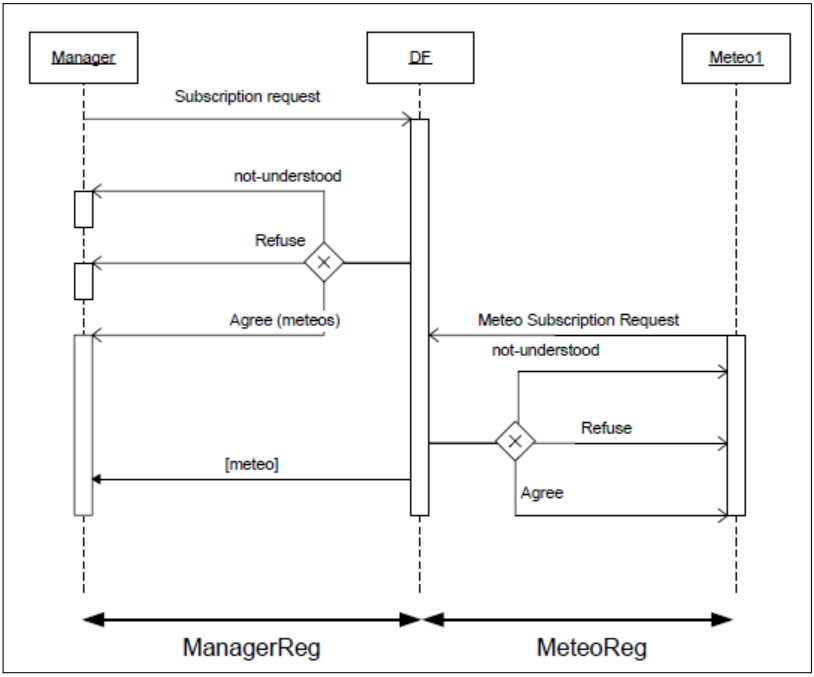

Figure 2.27: Interaction Protocol Example, (Tomás and Garcia 2005).

2.4.2 Works Using Different Approaches

Table 2.8 summarizes works that propose solutions to cope with the traffic control and emergency response issue using approaches

from other fields.

Title	Scenario	Field
Reducing Emergency Services Response Time in Smart Cities: An Advanced Adaptive and Fuzzy Approach (Djahel et al. 2015)	Adaptive traffic control system to enables faster emergency services response in smart cities	Adaptive and Fuzzy Approach
A heuristic implementation of emergency traffic evacuation in urban areas (Kang et al. 2013)	Traffic control under emergency situations	Algorithms and Graphs
Modeling the Impact of VANET-Enabled Traffic Lights Control on the Response Time of Emergency Vehicles in Realistic Large-Scale Urban Area (Noori 2013)	Traffic lights control to decrease the response time of emergency cars	Electronic V2V and V2I communication
Making Way for Emergency Vehicles at Oversaturated Signals under Vehicle-to-Vehicle Communications (Cetin and Jordan 2012)	Traffic signs control to improve emergency vehicle dislocation	Electronic V2V and V2I communication
Traffic organization method for emergency evacuation based on information centrality (Wu et al. 2010)	Traffic control under emergency evacuations	Optimization
Rapid Handling of Urban Traffic Emergencies Based on Decision and Command Support System (Lu 2009)	Handling urban traffic emergency	Software engineering
Model Reference Adaptive Control Framework for Real-Time Traffic Management under Emergency Evacuation (Liu et al. 2007)	Traffic control under emergency evacuations	Adaptive Control Theory

Table 2.8: Works from others approaches.

Among the cited works in Table 2.8, Shumin et al. 2010 proposed a solution that establishes a decision support system for urban emergency traffic control based on expert systems. They consider an expert system endowed with enough knowledge of the overall problem in order to be used to solve the respective problem.

According to Wooldridge 2009, such centralized problem solving approach is less encouraged to cope with complex problems. Since it assumes that a component has the appropriate expertise to decompose an overall problem to specific ones, thus, has the knowledge of the task structure. It is less sure to find a component that

has a global view of the environment as well as events occurring. The same main problem observed in Shumin et al. 2010's approach is also encountered in Lu 2009, and Kang et al. 2013.

Liu et al. 2007 integrate both dynamic network modeling techniques and adaptive control theory to manage the traffic at real-time during emergency evacuations. They consider that the traffic flow during emergency is full of uncertainties, therefore needs a dynamic modeling technique as well as a real time adaptive control.

Although such approach shows interesting results regarding optimization of traffic evacuation, it does not include explicitly the emergency responders management into the scenario, in the sense of several services cooperating according to a coordination mechanism in order to reach a same overall goal. Being the emergency responders management one of the important problems to consider in order to improve an emergency response process. The observation highlighted in Liu et al. 2007's approach is also encountered in Wu et al. 2010, and Kang et al. 2013.

2.4.3 Works Characteristics

Some of the works in the literature focus more on emergency management at and around the local of the incident than on traffic control. This may be due to several reasons. A particular reason can be the fact that in real emergency cases, the first and automatic reaction of people is to try to save a maximum of victims, managing the emergency into the incident local. In this way, to control adequately the traffic often turns to be less important, or even a future target.

Nevertheless, to consider the traffic control as one of the solutions of an emergency management can improve significantly the effectiveness of such management. Since the response process, which is the second stage of an emergency management, as presented in Section 2.3, begins with the displacement of emergency teams to the emergency location. Therefore, an adequate traffic control maximizes the probability of emergency vehicles to reach as fast as possible their destinations, hence to save a maximum number of lives.

Among works that focus on traffic control in emergency situations using MAS approaches, most of them do not present a clear cooperation or/and coordination mechanism. Some works do not have even one section or paragraph explaining clearly a cooperation technique presented in the literature, or a known or own coordination mechanism used in the work. Generally what they present

is a conceptualization of their system, followed by the system architecture with details of each agent function and some interaction protocol. The cooperation and coordination tend to be somewhat implicit, using each agent functions.

Furthermore, there are works in which the cooperation approach is quite mixed or sometimes confused with the coordination approach. As we present in Section 2.1.6 and 2.1.7, respectively, we consider the cooperation and coordination in MAS as two different elements, even though they are close. A cooperation consist in working together in order to solve something. Thus, a cooperation technique presents the way how to work together. Whereas a coordination consists in managing the interactivity between the agents. So a coordination mechanism presents the form of such management.

Finally, there are works that consider almost everything as agents in their representation of the MAS: from the traffic with it infrastructure, to the emergency teams and other personnel. Some others works even use the environment approach, but they still mix and/or confuse the endogenous environment with the exogenous environment.

Several authors may disagree with such observation, since a MAS, has to be considered as an agent at a higher level, as defended by Demazeau 1995 in his recursion principle. Others may affirm that environment and organization approaches are not obligatory in a MAS representation.

We still consider that in the case of this studied scenario, to distinguish MAS agents from traffic components, as well as MAS environment from the traffic system allows us to propose adequate strategies to each of the systems. That is, to propose a cooperation technique and coordination mechanism with an interaction protocol for agents at one side. At the other side, to propose a traffic control strategy for the traffic system.

3 PROPOSAL

3.1 OVERVIEW

In this dissertation, we study the problem of controlling the urban traffic system under an emergency incident as presented in Section 1.1. More specifically, we focus on the traffic lights control and the emergency vehicles management and routing during an emergency response process presented in Section 2.3. We propose the use of a MAS approach endowed with an emergency response strategy that controls the traffic lights and manages emergency vehicles in order to reach the objectives established in Section 1.2. Figure 3.1 depicts the overall system architecture of our proposal.

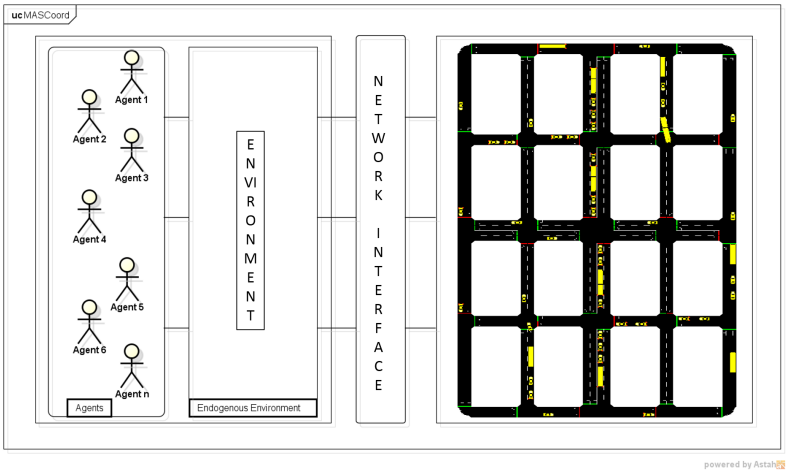

Figure 3.1: Overall system architecture.

A simple urban traffic network is depicted on the right side of Figure 3.1, representing the urban traffic system composed basically of: links, junctions, traffic lights and vehicles. There are also some emergency vehicle bases at different points of the traffic network. All these traffic elements are defined in Section 2.2.1. On the left side we have a representation of the MAS, composed of agents which interact one another using a coordinated strategy, as well as with the traffic system using an endogenous environment. The traffic system and the MAS are connected by a network interface which allows the exchange of data between both of the systems.

This section is divided in two main parts. The first part presents

the proposed emergency response strategy with respect to the emergency response process, as well as the traffic light control procedure. Some concepts used therein are also introduced. The second part presents the proposed MAS, its objectives and components, as well as a description of how this MAS manages the urban traffic system using the emergency response strategy presented in the first part of this section.

3.2 EMERGENCY RESPONSE STRATEGY AND TRAFFIC LIGHT CONTROL

This work follows the approach used by Borges et al. 2010 presented in Section 2.3, which considers the emergency management as a process that comprises three main stages: the notification and validation of an emergency incident, the emergency response process and the closing activity. The emergency response strategy proposed here focuses on the emergency response process and consists specifically of a systematic combination of actions that control traffic lights as well as managing and routing emergency vehicles in order to reach the objectives established in Section 1.2.

3.2.1 Some Concepts

Most of the elements and concepts regarding the urban traffic system and the emergency management are introduced in Chapter 2. Nevertheless, before presenting the proposed emergency response strategy and traffic light control, we define in this section some of these concepts in the context of this proposal. Some others new concepts used in this proposal are also introduced.

Definition 1. The term *responder*, denoted by the variable r, refers to an emergency vehicle from which we consider the following types: ambulances, fire-fighters and the police.

Definition 2. Each responder r has some preference to cross junctions compared to other vehicles. This preference is based on a priority degree $\delta_r \in \{1, 2\}$, according to the type of r and sometimes according to its travel destination as follow:

- $\delta_r = 2$ (*hight degree*) if r is an ambulance or if r is a fire-fighter going to the emergency location.
- $\delta_r = 1$ (*low degree*) if r is a fire-fighter returning to its base or if r is the police.

Definition 3. An *incoming link* is a link that approaches a junction. An *outgoing link* is a link that leaves a junction. A *prioritized link* is a link on which at least one responder is prioritized. The variables l, f_l and t_l denote respectively a link, the junction that l leaves and the junction that l approaches.

Definition 4. We use the term *junction* to refer to both the traffic junction as an intersection of links and the traffic light as an electronic component that controls such traffic junction. The variable j denotes a junction. I_j represents the set of incoming links to j and O_j represents the set of outgoing links to j. Considering Definition 3, $l \in O_{f_l}$ and $l \in I_{t_l}$.

Definition 5. A path, denoted by the variable p, is the way from which a responder travels during the emergency response process. A path is represented by a list of links between a origin junction and a destination one, and n_p denotes the number of junctions on p.

Definition 6. A *router*, denoted by a_{rtr}, is a agent that controls one responder. A *traffic light controller*, denoted by a_{tlc}, is a agent that controls one junction. Both of the agents are described in Section 3.3.2.2 and 3.3.2.3 respectively.

Definition 7. A *priority request* is a message that a a_{rtr} sends to a a_{tlc} to get priority at the junction controlled by a_{tlc}. The priority request procedure is introduced in Section 3.2.2.2.

Definition 8. A a_{tlc} *prioritizes* a responder r at a junction j when j assigns green light to the link $l \in I_j$ from which r approaches j.

Definition 9. A junction j is on *priority mode* when j is controlled by a a_{tlc}. Otherwise, it is on *conventional mode*.

3.2.2 Emergency Response Strategy

During the emergency response process, responders need to reach as fast and soon as possible their destinations in order to either assist victims at the emergency location or evacuate victims to health care locations. For this reason, (1) they must travel on a *best path* from their origin to their destination and (2) a_{tlc} agents have to prioritize them as soon as possible when requested.

Since there are several responders traveling on the traffic network, (3) the conflicts between priorities at junctions have to be managed. We also intend to minimize the impact of priorities on

the traffic flow. Therefore, (4) responders have to be prioritized according to their position in order to avoid that their priorities affect the traffic management at junctions far from them.

Furthermore, when a responder r is prioritized at some junction j, (5) vehicles that do not obstruct the crossing of r through j are allowed to cross j together with r. Finally, (6) junctions have to be reset immediately to conventional mode once all the responders have crossed them.

The emergency response strategy is described by steps that comply with the requirements (1) to (6) as follow:

1. *Best Path Determination*: when an emergency alert is sent, each responder r travels from its base to the emergency location using a path p determined by its a_{rtr}.

2. *Priority Request*: during the travel of each responder r through p, its a_{rtr} sends systematically some priority requests to a_{tlc} agents which control junctions on p, in order to prioritize r at their respective junctions.

3. *Priority Conflicts Management*: priorities at each junction j are managed according to the responder priority degrees δ_r and the distance between each r and j.

4. *Junction Reset*: junctions are reset immediately to conventional mode once all the responders have crossed them.

The *Best Path Determination*, *Priority Request* and *Priority Conflicts Management* are presented with more details in the sequence.

3.2.2.1 Best Path Determination

We approach the question of best path determination as a *single-pair shortest-path* problem described by Cormen 2009. Given a weighted directed graph $G = (J, L, w)$, where J denotes the sets of all junctions, L the set of all links and $w : L \to \mathbb{R}$ is a weight function mapping links $l \in L$ to real-valued weights. We want to find a shortest weighted path from a source $j_s \in J$ to a target $j_t \in J$. This proposal considers and asses two kinds of weights:

- *The distance* which refers to the length of the link.

- *The density* which refers to the number of vehicles by kilometer in the link.

To determine the shortest weighted path, we use the *Dijkstra* algorithm which solves the *single-source shortest-paths* problem on a weighted directed graph for the case in which all edge weights are nonnegative. Notice that both the distance and the density weights are nonnegative values, and the *single-pair shortest-path* is a specific case of the *single-source shortest-paths*.

3.2.2.2 Priority Request

Once the best path p is found, the agent a_{rtr}, responsible for the responder r, sends priority requests to a group of a_{tlc} agents during the routing of r on p, so that at least the $\alpha, (\alpha \in [1, n_p])$ next a_{tlc} agents prioritize r at junctions j on p. The value of α impacts on the objective of this dissertation. For instance, if $\alpha = 1$, the responder r is prioritized just in the next junction of its path p, likely limiting its overall speed. An example of some priority request with $\alpha = 1$ is depicted in Figure 3.2.

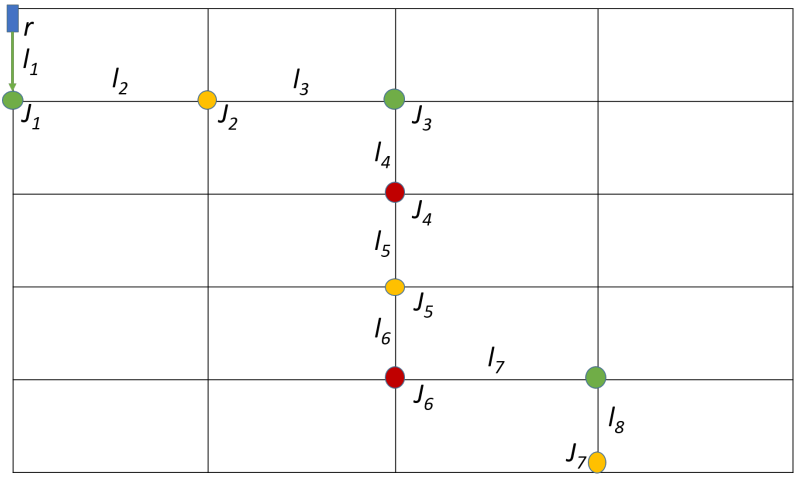

Figure 3.2: Example of priority requests with $\alpha = 1$.

If $\alpha = n_p$, all next junctions on p will prioritize r giving it a free way to the destination. However, the impact on the traffic flow will be significant. An example of some priority requests with $\alpha = n_p$ is depicted in Figure 3.3.

Algorithm 3.2.1 describes the priority request procedure used by a_{rtr}.

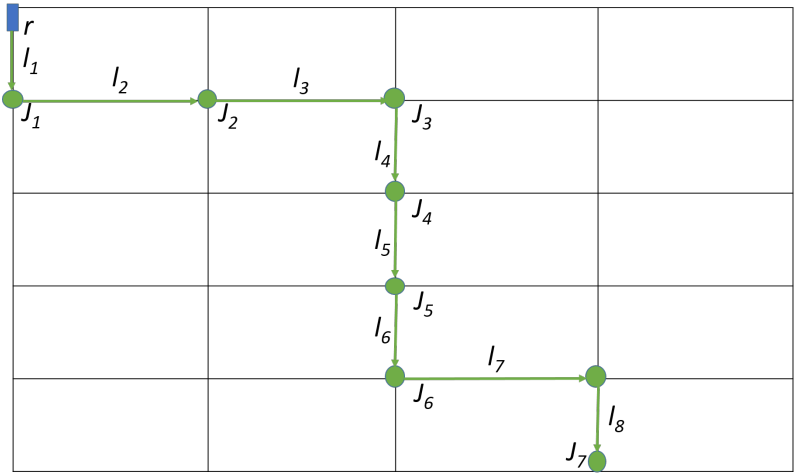

Figure 3.3: Example of priority requests with $\alpha = n_p$.

Algorithm 3.2.1: PRIORITYREQUEST(p, i, α)

comment: i is the index of the initial junction to which

comment: the request is sent.

comment: $p = [l_0, l_1, \ldots, l_n]$, f_l and t_l are junctions.

comment: (see Definition 3 in Section 3.2.1).

$c \leftarrow p.size$
if $i = 0 \ \wedge \ c \geq \alpha$
  for $k \leftarrow 0$ **to** $(\alpha - 1)$
    $l \leftarrow p[k]$
    send a request message to a_{tlc} controlling t_l
  $i \leftarrow \alpha$
  wait until r arrives at l
  PRIORITYREQUEST(p, i, α)

else if $i < c - (\alpha - 1)$
  for $k \leftarrow i$ **to** $i + (\alpha - 2)$
    $l \leftarrow p[k]$
    send a request message to a_{tlc} controlling t_l
  $i \leftarrow i + (\alpha - 1)$
  wait until r arrives at l
  PRIORITYREQUEST(p, i, α)

else if $i < c$
  for $k \leftarrow i$ **to** $c - 1$
    $l \leftarrow p[k]$
    send a request message to a_{tlc} controlling t_l
  $i \leftarrow i + (\alpha - 1)$
  wait until r arrives at l
  PRIORITYREQUEST(p, i, α)

else $finish$.

3.2.2.3 Priority Conflict Management

When more than one responder r coming from different links request some priority simultaneously, these priorities conflicts are managed. Each traffic controller agent a_{tlc} determines a priority co-

efficient λ_r for each request in order to determine the crossing preference through the junction j controlled by a_{tlc}. A priority request comprises the following elements:

- r is the responder for which the priority is requested.
- l is the link from which r approaches the junction j, $l \in I_j$.
- $d_{r,j}$ is the distance between r and j.
- δ_r is the r priority degree.

The priority coefficient λ_r is found considering the distance $d_{r,j}$ and the priority degree δ_r assigned to r. If $d_{r,j}$ is less than the length of the shortest link $l \in I_j$, λ_r corresponds to $d_{r,j}$. Otherwise, λ_r depends on $d_{r,j}$ and δ_r. The operation is formulated as follow:

$$\lambda_r = \begin{cases} \delta_r, & \text{if} \quad d_{r,j} \leq \min(\theta_{l_1}, \theta_{l_2}, \ldots, \theta_{l_n}), l_i \in I_j, \theta_{l_i} \\ \text{is the length of } l_i & \\ \\ \frac{\delta_r}{d_{r,j}} & \text{otherwise.} \end{cases} \tag{3.1}$$

The priority is assigned to the responder r with the greatest λ_r. When a responder r from a link l is prioritized, every responder r' on l has the right to cross the junction simultaneously at the same junction signal. The next priority assignment occurs only after all the responders on l cross the junction.

3.2.3 Traffic Light Control

The proposed traffic signal control is based on guidelines defined by DENATRAN 2014 and is employed only during an emergency response process. We assume that all junctions in the traffic network may be controlled. Junctions in the traffic network can operate in conventional and priority modes, as introduced in Section 3.2.1. When the junctions are in conventional mode, they are controlled by a fixed time traffic signal plan. We follow the time used in DENATRAN 2014, Chapter 5, Figure 5.3. We define in the sequence the control characteristics, objectives and the main elements used for such control strategy.

3.2.3.1 Control Characteristics

The control is totally actuated, i.e., the time of junction stages is based on the responders demand. The control is isolated, in which each junction is managed independently of the others. However, isolated control can affect seriously the traffic flow performance in situations where junctions are close. Coordination is left as future work. The control is centralized, connected to a central (group of) machine(s) where the MAS runs.

3.2.3.2 Control Objectives

When a junction j is on priority mode, the control strategy has two objectives:

1. assign green light to a prioritized link $l \in I_j$.
2. allow vehicles from links $l' \in I_j$ and $l' \neq l$, which do not obstruct the crossing of vehicles from l through j to cross j simultaneously.

The set of stages S_j for each j is determined considering these two objectives.

3.2.3.3 Set of Stages

To determine the set of stages S_j of some junction j, we use the diagram of conflicts which consist on the schematic representation of a junction geometry, considering all the vehicular movements that can occur therein. From this diagram, we derive a table of conflicts that shows conflicting movements into j area.

According to the conflicts table approaches and given a junction j, let us consider and define some elements that will help us to determine the set of stages S_j, based on DENATRAN 2014, chapter 5.

- M_V_j is the set of vehicular movements through j. $M\_V_j = \{m\_v_1, m\_v_2, \ldots, m\_v_n\}$.
- S_V_j is the set of signs controlling vehicular movements through j. $S\_V_j = \{s\_v_1, s\_v_2, \ldots, s\_v_n\}$ and s_v_i controls m_v_i.
- $s\_v_i \in \{g, y, r\}$, where g is the green sign, y is the yellow sign and r is the red sign.

- G_M_l is a group of vehicular movements $m\_v \in M\_V_j$ from a same link $l \in I_j$.
- G_S_l is a group of signs $s\_v \in S\_V_j$ controlling a group of vehicular movements G_M_l, $l \in I_j$.
- We say G_S_l is *green* $\iff \forall s\_v \in G\_S_l, s\_v = g$.

Considering the elements defined above, we define the property 3.2 as follow:

$$\forall l \in I_j, \exists s \in S_j \mid G\_S_l \textit{ is green} \tag{3.2}$$

The property 3.2 means that for each link $l \in I_j$ approaching a junction j, there is a stage $s \in S_j$ that assigns green light to all vehicular movements on l. This property ensures that when some priority for some responder r approaching j via a link $l \in I_j$ is requested, there is a stage $s \in S_j$ that allows r to cross j independently of its vehicular movement.

Let us define the relation $R : S\_V_j \rightarrow S\_V_j$ such that $\forall s\_v_i \in S\_V_j, s\_v_k \in S\_V_j, s\_v_i \; R \; s\_v_k \iff m\_v_i$ does not conflict with m_v_k. From R and given a link $l \in I_j$ such that G_S_l is *green*, we define the property 3.3 as follow:

$$\forall s\_v \in G\_S_l, \forall s\_v' \in S\_V_j, s\_v' \notin G\_S_l, s\_v \; R \; s\_v' \rightarrow s\_v' = g \tag{3.3}$$

The property 3.3 means that when two vehicular movements m_v_1 and m_v_2 do not conflict each other, the signs s_v_1 and s_v_2 controlling them respectively can be green at the same time. This property implies that when vehicles from a link $l \in I_j$ are crossing a junction j, vehicles from other links $l' \in I_j$ which vehicular movements do not conflict with vehicles from l are allowed to cross j simultaneously.

Consequently, for each junction j in the traffic network, the set of stages S_j is determined defining stages that comply with the properties 3.2 and 3.3. The green time of stages on a junction j in priority mode corresponds to the time required for all the responders cross j.

3.3 MULTI AGENT SYSTEM MODEL

The proposed MAS uses the emergency response strategy proposed in Section 3.2.2 to control junctions and manage and route responders. The MAS controls junctions and routes responders by

a_{tlc} agents, which uses *TLCTools* artifacts to interact with the traffic system, and a_{rtr} agents through *RTRTools* artifacts, respectively. Both of the agents were introduced in section 3.2.1. Responders are managed by both the *traffic analyzer* which interacts with the traffic system through the *ANL Tools* artifact, and a_{rtr} agents. The agent and environment elements are described with more details in the sequence. Figure 3.4 shows the overall system on the MAS view.

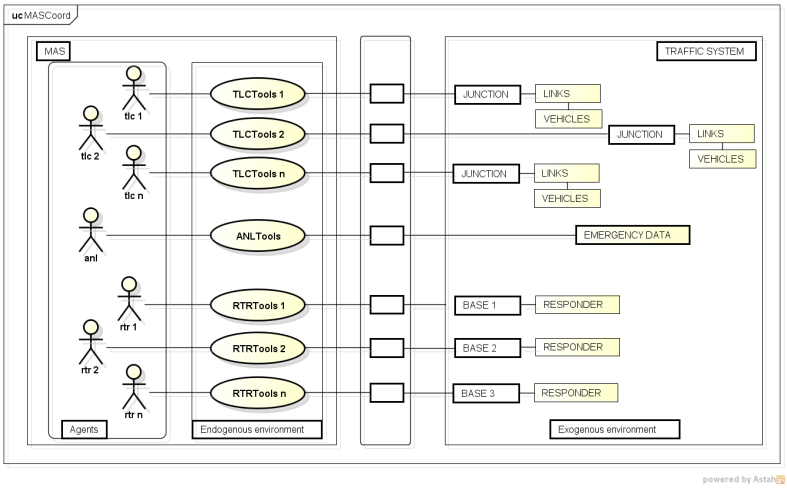

Figure 3.4: System Architecture.

The left side in Figure 3.4 shows a representation of the MAS with its two main levels: the agent level having the a_{anl}, a_{rtr} and a_{tlc} agent types, and the endogenous environment level with the artifacts used by each agent respectively. The right side represents the traffic system with its respective components, perceived by the MAS as an exogenous environment. In the middle, a network interface connects both of the systems allowing the exchange of data between the MAS artifacts and the traffic components.

This section is divided in three parts: we present first the MAS goal overview represented by a tree of goals and detailed by scenarios which describes some expected behaviors of the system in determined moments. The agents are described in the sequence, detailing their roles, missions and the endogenous environment through which they perceive and act on the traffic system. Lastly, the interaction protocols used by the agents in order to cooperate and coordi-

nate them selves are presented. Most of the concepts and elements defined in this section are presented following the Prometheus AE-Olus MEthod 2013.

3.3.1 Goal Overview and Scenarios

The MAS goal overview shown in Figure 3.5 is represented by a tree of goals divided in two main subgoals which represent the emergency response strategy. The *manage traffic* goal implies to create a_{tlc} agents according to the number of junctions, as well as to act on such junctions. In the same way, the *manage responders* goal implies to create routers that will control the emergency responders once an emergency is perceived. The number of a_{rtr} agents depends on the number of responder vehicles involved in the emergency.

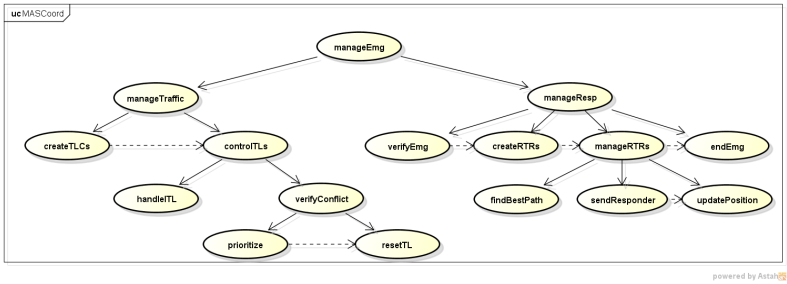

Figure 3.5: MAS Goal Overview.

3.3.1.1 Manage Emergency Scenario

The present scenario refers to the *manage emergency* goal which is the main MAS goal, and describes in broad outlines what happen into the MAS during the emergency response management, from the MAS start until the end of the emergency. Therein, we highlight some general perceptions, actions, interactions, objectives and scenarios generated by this one into the MAS. Table 3.1 summarizes the *manage emergency* scenario.

3.3.1.2 Manage Responders Scenario

This scenario describes how responders are managed by a_{rtr} agents during the emergency response process. It starts when a_{rtr}

Manage Emergency	
Description	This scenario presents the main steps of the MAS concerning the emergency response management
Trigger	MAS start
Steps	*Perception: get junction ids *Objective: create a_{tlc} agents *Objective: verify emergency occurrence *Perception: get emergency local and number of victims *Objective: create a_{rtr} agents *Interaction: send an emergency alert to a_{rtr} agents *Scenario: manage responders *Scenario: manage junctions *Objective: end emergency

Table 3.1: Manage Emergency Scenario.

agents are created and finishes with the end of the emergency response process. Table 3.2 summarizes the *manage responders* scenario.

Manage Responders	
Description	This scenario presents the main steps of the a_{rtr} agent concerning the responder management and routing
Trigger	a_{rtr} agents created
Steps	*Perception: perceive emergency data *Action: find the best path *Objective: generate α path segments *Interaction: request a priority to α first a_{tlc} *Objective: send responder *Objective: update position *Perception: r arrives at $\alpha - 1^{th}$ link l *Interaction: request a priority to more $\alpha - 1^{th}$ a_{tlc}. ... *Action: end routing *Perception: emergency finished *Action: end management

Table 3.2: Manage Responders Scenario.

3.3.1.3 Manage Traffic Lights Scenario

This scenario describes how junctions are managed by a_{tlc} agents during the emergency response process. It begins when a_{tlc} agents are created and finishes with the end of the emergency response process. Table 3.3 summarizes the *manage junctions* scenario.

Manage Junctions	
Description	This scenario presents the main steps of the a_{tlc} agent concerning the traffic light control
Trigger	a_{tlc} agents created
Steps	*Perception: get junction data. *Perception: receives a priority request for r at j *Objective: handle junction (for each r, determine λ_r) *Objective: verify conflicts *Objective: prioritize r *Perception: r crossed j *Objective: reset junction *Perception: emergency finished. *Action: end control

Table 3.3: Manage Traffic Lights Scenario.

3.3.2 Agents

As introduced before, the proposed MAS comprises three kinds of agent: a traffic analyzer a_{anl}, a a_{rtr} and a a_{tlc}. There are three types of a_{rtr} agent: ambulance, firefighter and police, denoted amb, frf and plc respectively. Each agent perceives and acts on the endogenous environment which has three kinds of artifacts: *ANL Tools*, *RTRTools* and *TLCTools* used by the a_{anl}, a_{rtr} and a_{tlc} agents respectively. The agents and the endogenous environment are presented in the sequence.

3.3.2.1 Traffic Analyzer Agent (a_{anl})

The *Traffic Analyzer Agent* is responsible for creating the a_{tlc} agents, since their number corresponds to the number of junctions. It also creates a_{rtr} agents once an emergency occurs, according to the number of responder vehicles involved. Lastly, it is responsible for managing the response process from sending a_{rtr} agents, verifying the number of victims not assisted yet, delegating ambulances for evacuation, until to inform responders of the end of the emergency so that they can go back to their respective bases. Table 3.4 summarizes the a_{anl} agent description.

Traffic Analyzer Agent (a_{anl})	
Description	The agent responsible for verifying the emergency occurrence, stage, data, and end, as well as creating the a_{tlc} and a_{rtr} agents
Cardinality	1

Table 3.4: Traffic Analyzer Agent.

3.3.2.2 Traffic Router Agent (a_{rtr})

The *Traffic Router Agent* is responsible for routing the responder vehicle from a given origin O to a destination D informed by the a_{anl}. It finds the best path p from O to D, and sends systematically a priority request to a_{tlc} agents on p. After a routing to the emergency local, ambulance a_{rtr} agents evacuate immediately victims to a health care locals and wait for an a_{anl} requirement. Table 3.5 summarizes the a_{anl} agent description.

Traffic Router Agent (a_{rtr})	
Description	The agent responsible for controlling and routing the responders. It finds the best path and guide the vehicle
Cardinality	corresponds to the number of responder vehicles.

Table 3.5: Traffic Router Agent.

3.3.2.3 Traffic Light Controller Agent (a_{tlc})

The *Traffic Light Controller Agent* is responsible for controlling some junction, prioritizing responders when requested, handling conflict of priorities, and resetting the junction to the conventional control after the priority assignment. Table 3.6 summarizes the a_{tlc} agent description.

Traffic Light Controller Agent (a_{tlc})	
Description	The agent responsible for controlling some junction, prioritizing the responders and handling the conflicts between the their priority requests
Cardinality	corresponds to the number of junctions

Table 3.6: Traffic Light Controller Agent.

3.3.3 Environment

The MAS environment is composed of three artifacts: *ANLTools*, *RTRTools* and *TLCTools*, used by a_{anl}, a_{rtr} and a_{tlc} agents respectively, as resources or features for their activities

- *ANLTools* are used to get junction ids and the emergency location, the number of victims, and to save emergency data to be used for the response process assessment.

- *RTRTools* are used to compute the best path, to send responders and to get the responder position along the route;
- *TLCTools* are used to get the junction data, to compute the set of stages to set signal on junctions, and to reset signals.

3.3.4 Organization

We adopt the *Moise* Organization Modeling Language (OML), (Hubner et al. 2007), to specify the MAS organization. *Moise* OML considers three dimensions to define a MAS organization.

3.3.4.1 Structural Dimension

The structural dimension specifies roles, links and groups in the organization. Roles are used to assign constraints on the behavior of agents playing it. Links establish relations between roles. Groups comprise roles, links and define formation constraints between roles.

According to this perception, we define first the roles in the organization. These roles correspond to the agent types defined in the MAS:

- *The Traffic Analyzer* represented by a_{anl}.
- *The Traffic Router* represented by a_{rtr}.
- *The Traffic Light Controller* represented by a_{tlc}.

Following, links are specified between the roles defined.

- A bi-directional communication from a_{anl} to a_{rtr}.
- A bi-directional communication from a_{rtr} to a_{tlc}.

In the formation constraints specification the compatibilities are established between the two links defined above.

3.3.4.2 Functional dimension

The functional dimension specifies goals, missions and schemes in the organization. A mission is a set of goals to be assigned to roles within norms. A scheme is a global goal decomposition tree assigned to a group.

In this proposed MAS model, the global goal decomposition tree in the scheme corresponds to the goals tree shown in Figure 3.5 in Section 3.3.1. Missions comprises goals defined in the goal tree and are defined as follow:

- $m1$: Verify the emergency occurrence, create a_{rtr} and a_{tlc} agents, manage a_{rtr} agents and ends the emergency response process.
- $m2$: Find the best path, send the responder and update the responder position.
- $m3$: Handle the junction, verify conflicts, prioritize responders and reset the junction.

3.3.4.3 Normative Dimension

The normative dimension specifies norms which consists of obligations, permissions and interdictions in the organization. A norm associates an authority to a role toward a mission. Such authority can be an obligation, permission or interdiction.

In our proposed MAS model, we define all the norms with the *obligation* type, as we use the coordination through join intention mechanism to handle the cooperation between the agents. So, the defined norms serve as a convention to the commitment of each agent to its respective role. These norms are specified as follow:

- $n1$: Obligation for a_{anl} role to accomplish the mission $m1$.
- $n2$: Obligation for a_{rtr} role to accomplish the mission $m2$.
- $n3$: Obligation for a_{tlc} role to accomplish the mission $m3$.

3.3.5 MAS Cooperation

To make agents working together in our proposed MAS model, we adopt the task sharing cooperation technique introduced in Section 2.1.6.1. We begin decomposing our overall problem into smaller sub problems recursively until each sub problem to be solved by an individual agent. Following, each agent develops plans to solve its own problem. Agents share information useful for solving their respective problems. Finally, the sub problems solutions are combined and result to the overall solution. Each step are described in detail in the sequence.

3.3.5.1 Problem Decomposition

Our overall problem is to manage the emergency response process. We decompose this problem into three sub problems: (1) To control junctions, (2) To manage Responders and (3) To route responders. Although each sub problem could be decomposed into smaller problems, we consider these sub problems enough to be solved by individual agents. Figure 3.6 shows the problem decomposition tree, assigning each sub problem to its agent solver.

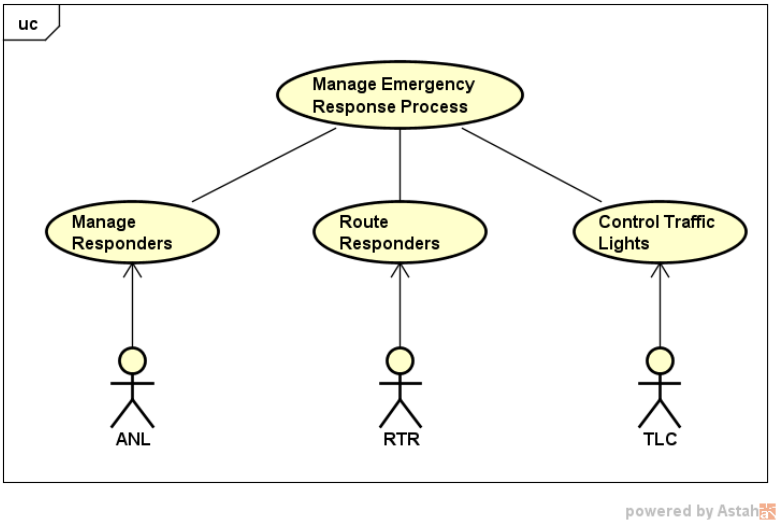

Figure 3.6: Problem Decomposition Tree.

3.3.5.2 Sub Problems Solution

The *manage responder* sub problem is solved by the a_{anl} agent by verifying when an emergency happen, handling the emergency evolution and ending the emergency response process once it verifies that all the objectives are reached by the a_{rtr} agents. The a_{anl} handles the emergency evolution by centralizing all the information that indicate the activities progress. Information such as the number of victims not assisted yet, the responders traveling and the responder that always finished their activities.

The *route responders* sub problem is solved by the a_{rtr} agent, which finds the best path p to be traveled by the responder r routed

by it. Once p is found, a_{rtr} sends r and updates r position sending such position to all tls agent on p.

The tls agent solves the *control junction* sub problem by setting the traffic signs according to a traffic signal plan, verifies the conflicts between responders priorities being requested by a_{rtr} agents, prioritizes responders and reset the junction.

During the sub problem solution stage, the agents exchange information in order to solve their respective subproblems. These interactions are described and handled by protocols introduced in Section 3.3.7. Figure 3.7 shows the sub problem solutions tree.

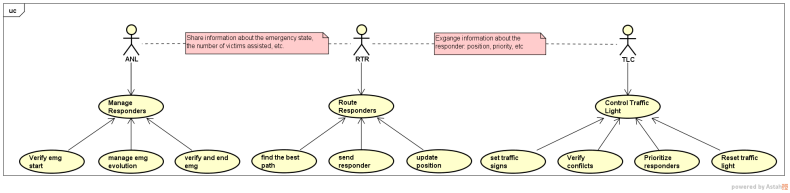

Figure 3.7: Sub Problems Solution Tree.

3.3.5.3 Solution Synthesis

This is the last stage of the task sharing cooperation technique. In this stage, the solutions found by the a_{anl}, a_{rtr} and tls agents are merged in order to solve the overall problem. Figure 3.8 shows the solution synthesis tree composed by the overall problem as the root, followed by the sub problems, followed by the solutions to these sub problems, and ended by the agents responsible for solving implementing these solutions.

3.3.6 MAS Coordination

To coordinate the agent cooperation in our proposed MAS model, we use the *join intention* coordination mechanism introduced in Section 2.1.7.2, which consists of managing cooperative activities between agents through a joint commitment to an overall aim and their individuals commitments to the specific tasks that they have been assigned. We use the MAS organization approach to implement the coordination through join intention.

Each agent when created, assumes a role defined in the organization. Consequently, the agent commits to the goals that com-

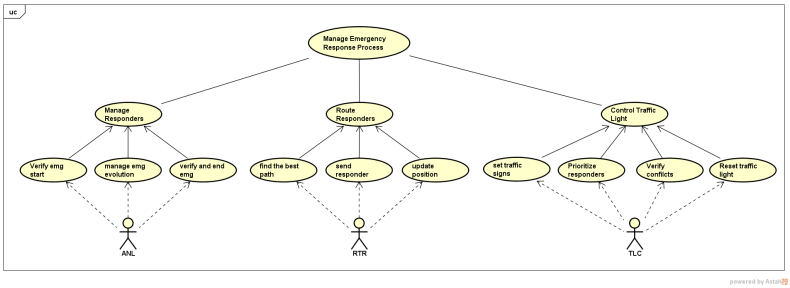

Figure 3.8: Solution Synthesis Tree.

pose the mission assigned to the respective role. The norms defined in the organization serve as a convention to the agent commitments. Since they oblige the agents to accomplish the mission assigned to their respective roles.

3.3.7 Interaction Protocols

The agents interact each one to others through two proposed protocols. The *ambulance Response Interaction protocol*, denoted by *RRP*, involves the a_{anl}, a_{amb} and a_{tlc} agents. The *Router Response Interaction Protocol*, denoted by *ARP*, occurs among the a_{anl}, a_{tlc} and the other a_{rtr} agents. Both interaction protocols start when the a_{anl} agent sends an emergency *alert* message to a_{rtr} agents and have seven events that trigger the sending of messages, as well as they have the same messages composing the routing process. They differ one from another on the steps and messages that follow the routing process. These protocols are detailed as follow.

3.3.7.1 Router Response Interaction Protocol (RRP)

In the Router Response Interaction Protocol, when a plc or frf finishes a routing to the emergency location, it acts in the emergency location until all the victims are assisted. After the emergency finishes, the a_{anl} agent sends an *endEmg* message to a_{rtr} which returns to its base and sends to a_{anl} a *finish* message. Figure 3.9 shows the Router Response interaction protocol.

The numbers before messages in Figure 3.9 refers to events that trigger the sending of such messages. These events are:

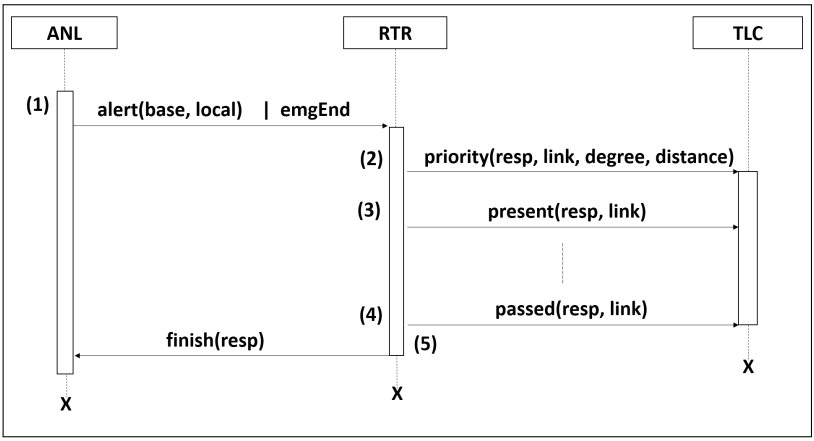

Figure 3.9: Router Response Interaction Protocol.

- **(1)**: the a_{anl} agent perceives an emergency occurrence or the emergency response process finishes.
- **(2)**: the a_{rtr} agent found the best path and starts sending priority requests.
- **(3)**: the a_{rtr} responder r arrives on the link $l \in I_j$, where j is the junction controlled by the a_{tlc} agent to which the priority request message was sent.
- **(4)**: the a_{rtr} responder r passed j, the junction controlled by the a_{tlc} agent to which the priority request message was sent.
- **(5)**: the a_{rtr} finished the routing.

3.3.7.2 Ambulance Response Interaction Protocol (ARP)

In the Ambulance Response Interaction Protocol, when an amb agent finishes a routing to the emergency location, it evacuates victims to a health care location. After, it sends a *finish* message to a_{anl}. If there are victims not assisted yet, the a_{anl} replies a *get victim* message to amb which restarts the protocol. Figure 3.10 shows the ambulance response interaction protocol.

The numbers before messages in Figure 3.10 represents events that trigger the sending of such messages. These events are:

- **(1)**: the a_{anl} agent perceives an emergency occurrence.

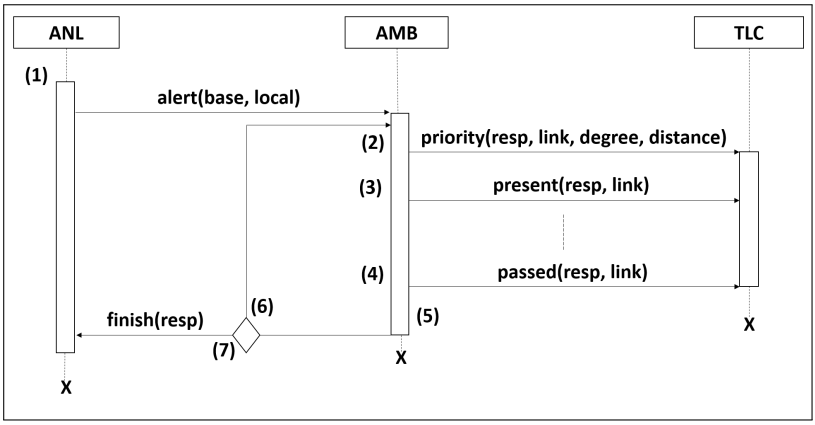

Figure 3.10: Ambulance Response Interaction Protocol.

- **(2)**: the amb agent found the best path and starts sending priority requests.
- **(3)**: the amb responder r arrives on the link $l \in I_j$, where j is the junction controlled by the a_{tlc} agent to which the priority request message was sent.
- **(4)**: the amb responder r passed j, the junction controlled by the a_{tlc} agent to which the priority request message was sent.
- **(5)**: the amb finished the routing.
- **(6)**: if the amb has just get the victim in the emergency location, it evacuates such victim to an health care location.
- **(7)**: if the amb is at the health care location, it finishes the routing.

3.3.7.3 Interaction Messages

Each interaction message mentioned both in the *ARP* and the *RRP* protocols are described in the sequence, following the Prometheus AEOlus MEthod 2013. They are presented in the same order of their occurrence in the MAS.

The *alert* message is the first interaction message sent from the a_{anl} agent to a_{rtr} agents informing them about some emergency occurrence. This message is triggered once the a_{anl} agent receives

an emergency notification from the traffic system. Table 3.7 summarizes the *alert* message.

Alert	
Description	The a_{anl} agent informs a_{rtr} agents about a traffic emergency occurrence
Target	*tell*
Content	alert(base, local)
Variables	*base: the responder base *local: the emergency location

Table 3.7: Alert Message.

The *priority* message is the second interaction message sent from a_{rtr} agents to a_{tlc} agents requesting the priority of their responder at junctions controlled by this a_{tlc} agents. This message is triggered once the a_{rtr} agent found the *best path* through which its responder will travel. Table 3.8 summarizes the *priority* message.

Priority	
Description	The a_{rtr} agent informs the a_{tlc} agents about its responder priority
Target	*tell*
Content	priority(a_{rtr}, link, degree, distance)
Variables	*rtr: the a_{rtr} agent that requests the priority *link: the link l from which the responder r approaches the junction j *degree: the priority degree *distance: the distance between r and j

Table 3.8: Priority Message.

The *present* message is the third interaction message sent from an a_{rtr} agent to some a_{tlc} agent informing that its responder r is present on the link $l \in I_j$, j controlled by a_{tlc}. This message is triggered once r enters on l. Table 3.9 summarizes the *present* message.

The *passed* message is the fourth interaction message sent from an a_{rtr} agent to some a_{tlc} agent informing that its responder r passed the junction j controlled by a_{tlc}. This message is triggered once r passed j. Table 3.9 summarizes the *passed* message.

The *finish* message is the fifth interaction message sent from an a_{rtr} agent to the a_{anl} agent informing the end of the routing. This message is triggered once the a_{rtr} responder r finishes the trip. Table 3.11 summarizes the *finish* message.

The *endEmg* message is the last interaction message sent from the a_{anl} agent to a_{rtr} agents informing them the end of emergency

Present	
Description	The a_{rtr} agent informs a_{tlc} agents that its responder r is present on the link $l \in I_j$, j controlled by a_{tlc}
Target	*tell*
Content	present(a_{rtr}, link)
Variables	*rtr: the a_{rtr} agent that requests the priority *link: the link l from which the responder r approaches the junction j

Table 3.9: Present Message.

Passed	
Description	The a_{rtr} agent informs a_{tlc} agents that its responder r passed the junction j controlled by a_{tlc}
Target	*tell*
Content	passed(a_{rtr}, link)
Variables	*rtr: the a_{rtr} agent that requests the priority *link: the link l from which the responder r passed the junction j

Table 3.10: Passed Message.

Finish	
Description	The a_{rtr} agent informs the a_{anl} agent that it finishes the routing
Target	*tell*
Content	finish(a_{rtr})
Variables	*rtr: the a_{rtr} agent

Table 3.11: Finish Message.

response process. This message is triggered once the all the victims are transported to health care locations. Table 3.12 summarizes the *endEmg* message.

EndEmg	
Description	The a_{anl} agent informs a_{rtr} agents the end of the emergency response process
Target	*tell*
Content	endEmg

Table 3.12: EndEmg Message.

3.4 IMPLEMENTATION

The implementation of this proposal involves basically implementing the urban traffic system, the MAS and the network interface, which are the tree main elements that compose the proposed overall system, as depicted in Figure 3.1, in Section 3.1. The implementation of each of these elements is detailed in the sequence.

3.4.1 Urban Traffic System

We use SUMO associated with the Object Oriented Programming Language *Java* to build the urban traffic system and implement the dynamic of different traffic components therein, such as traffic lights, vehicles, etc. We begin modeling the traffic network by specifying *nodes* which represent traffic junctions, *edges* which represent traffic links, and types of edges which specify some characteristics of links. These elements are specified in eXtensible Markup Language (XML) files named *nodes.nod.xml*, *edges.edg.xml* and *types.typ.xml* respectively.

Following, we specify in the traffic network configuration file *network.net.cfg*, the input files that SUMO will use to generate the traffic network as well as the output file with the network elements and values. The *network.net.cfg* file is depicted in Figure 3.11.

```
1 <configuration>
2    <input>
3        <edge-files value="edges.edg.xml"/>
4        <node-files value="nodes.nod.xml"/>
5        <type-files value="types.typ.xml"/>
6    </input>
7    <output>
8        <output-file value="network.net.xml"/>
9    </output>
10   <processing>
11       <no-turnarounds value="true"/>
12   </processing>
13 </configuration>
```

Figure 3.11: Example of a SUMO traffic network configuration.

Once the *network.net.xml* file is generated, we specify traffic routes and vehicles in the *route.rout.xml* file. The traffic flow of each route and the vehicles types are also specified therein.

To interact with the traffic system in SUMO, we use the *Traffic Control Interface for Java (TraCI4J)* library implemented using *Java*, which provides a real time interaction with a simulation in SUMO through the SUMO *Traffic Control Interface (TraCI)*. More details about TraCI4J can be found in the TraCI4J documentation.[1] Figure 3.12 depicts an implementation example of the connection with SUMO using TraCI4J.

```
30    /**Emmanuel K.D - init the Traffic simulation
31     * Receive the SUMO config file (.sumocfg)
32     * Receive the SUMO GUI Path
33     * and the random seed number and create a connection
34     * @throws InterruptedException
35     * @throws IOException */
36    public TrafficSim(String sumoCfgFile, String sumoGUIPath, String sumoExeProperty,
37            int randomSeed, boolean quitOnFinish) throws IOException, InterruptedException {
38        super(sumoCfgFile, randomSeed);
39        System.setProperty(sumoExeProperty, sumoGUIPath);
40        if(quitOnFinish) addOption("quit-on-end", "1");
41        runServer();
42    }
```

Figure 3.12: Example of a connection with SUMO using TraCI4J.

3.4.2 Multi Agent System

We implemented each level (agents, environment and organization) of the MAS proposed in this Chapter using *JaCaMo*. The implementation of each of these elements is detailed in the sequence.

3.4.2.1 Agents

We use *Jason* programming language to implement agents. Each agent type, i.e. a_{anl}, a_{rtr} and a_{tlc} is implemented in a file named *anl.asl*, *rtr.asl* and *tlc.asl* respectively. When an agent is created, the file corresponding to its type is instantiated. An implementation example of the a_{rtr} creation using *jason* is depicted in Figure 3.13.

Agents in *Jason* have three main elements: beliefs, goals and plans. A belief represents an information that the agent starts to accepts as true from some moment and keep in its mind, also named *belief base*. A belief can be included in the belief base when the agent

[1] <https://github.com/egueli/TraCI4J>

```
56 +!createRTRs(LCL, VTM) <-
57     +victims(VTM);
58     for(rtr(TYP,GRP,BSE,VTM,NUM)) {
59         for(.range(IDX,0,NUM-1)) {
60             .concat(TYP,GRP,IDX,NME);
61             .create_agent(NME,"/rtr.asl",[agentArchClass("c4jason.CAgentArch")]);
62             .send(NME,tell,alert(TYP,BSE,LCL,VTM));
63             !addRouter(NME,TYP);
64         };
65         !victimsToCarry(TYP,NUM);
66     };
67     .
```

Figure 3.13: Example of a_{rtr} creation.

is created or added from a perception in the environment or during some action. A belief can be removed from the belief base. Line 57 in Figure 3.13 shows an example of the number of victims being added in the $a + anl$ belief base.

Goals in *Jason* represent states that agents want to achieve in a given moment. Agents can have goals from their creation or can add a new goal during the realization of a given goal. Lines 63 in Figure 3.13 shows the examples some goal which consists to add router agents already created, passing their names and types, in the a_{anl} agent belief base.

Plans in *Jason* encapsulate actions that achieve agent goals. A plan is often triggered by some event which can be an environment perception or a given condition. The example of a plan to create a_{anl} agents is depicted in 3.13

3.4.2.2 Environment

The implementation of the MAS environment involves to implement each of the artifacts used by the agents. In the case of this proposal, we have the *ANLTools*, *RTRTools* and *TLCTools*. We use *Cartago* for this implementation. With *Cartago* an artifact corresponds to a *Java* class which extends another *Java* class named *Artifact*. An artifact has methods with *@OPERATION* annotation that turn these methods available to agents as actions. An artifact method with *@OPERATION* annotation is depicted in Figure 3.14.

3.4.2.3 Organization

The implementation of the MAS organization consists to specify each of the organization dimensions in a *XML* file. That is, the structural, functional and normative specifications. Figure 3.15 de-

```
@OPERATION
public void findPath(Object source, Object target) {
    try {
        TrafficGraph graph = new TrafficGraph();
        List<TEdge> edges = graph.gettEdges();
        for(TEdge e : edges) {
            double d = XMLManager.getLength(TRAFFIC_NET_FILE, e.getId());
            int nv = rmi.getNumVehicle(e.getId());
            graph.setWeight(e.getId(), d);                  //distance weight.
            graph.setWeight(e.getId(), nv/d);               //density weight.
        }
        TNode na = graph.getNode(""+source);
        TNode nb = graph.getNode(""+target);
        List<TNode> path = Dijkstra.getShortestPath(graph, na, nb);
        List<jason.asSyntax.Term> aux = new ArrayList<Term>();
        for(int i = 0; i < path.size()-1; i++) {
            TEdge e = graph.getEdge(path.get(i), path.get(i+1));
            aux.add(ASSyntax.createString(e.getId()));
            if(hasObsPropertyByTemplate("link", e.getFrom(), null, null, null))
                getObsPropertyByTemplate("link", e.getFrom(), null, null, null)
                    .updateValues(e.getFrom(), e.getTo(), e.getId(), e.getWeight());
            else defineObsProperty("link", e.getFrom(), e.getTo(), e.getId(), e.getWeight());
        }
        if(hasObsProperty("path")) getObsProperty("path").updateValue(ASSyntax.createList(aux));
        else defineObsProperty("path", ASSyntax.createList(aux));
```

Figure 3.14: Example of an artifact operation.

picts an example of the structural specification in the MAS organization.

```
    <structural-specification>
        <role-definitions>
            <role id="anl" />
            <role id="rtr" />
            <role id="tlc" />
        </role-definitions>

        <group-specification id="group">
            <roles>
                <role id="anl" min="1" max="1" />
                <role id="rtr" />
                <role id="tlc" />
            </roles>

            <links>
                <link from="anl" to="rtr" type="communication" scope="intra-group"
                    bi-dir="true" />
                <link from="rtr" to="tlc" type="communication" scope="intra-group"
                    bi-dir="true" />
            </links>

            <formation-constraints>
                <compatibility from="anl" to="rtr" type="compatibility" />
                <compatibility from="rtr" to="tlc" type="compatibility" />
            </formation-constraints>
        </group-specification>
    </structural-specification>
```

Figure 3.15: Example of the structural specification in the MAS organization.

3.4.2.4 Interaction

The implementation of the interaction in MAS consists into the implementation of messages sent between the agents, which compose the interaction protocols presented in Section 2.1.5. With Jason, these messages are implemented in the agent files. Figure 3.16 shows the example of a priority request sent from a a_{rtr} to an a_{tlc} agents.

```
65 +!sendAlert(RTR,SRC,LK,DRG) : link(ANT,TL,LK,D0) <-
66     distance(SRC,ANT,DST);
67     .send(TL,tell,priority(RTR,LK,DRG,DST+D0));
68     .
```

Figure 3.16: Example a priority request implementation.

3.4.3 Network Interface

We implement the network interface using the Java *Remote Method Invocation (RMI)*. The traffic system acts as a server, responding requests made by MAS environment artifacts, as clients. Line 57 in Figure 3.14 shows the example of a request made by *RTRTools* via *RMI*, in order to get the number of vehicles on a given link.

4 ASSESSMENT

4.1 OVERVIEW

We focus our assessment on the effectiveness of our proposal in reaching the aim defined in Section 1.2, by achieving the two objectives established therein. In this way, we model first an urban traffic network in Section 4.1.1, and simulate an emergency incident occurring in this traffic network, in Section 4.1.2. We also define in Section 4.1.3 some metrics to be used as parameters for the assessment. Following, we determine in Section 4.2, the value of α to be used in the simulation. Lastly, we present the assessment results in Section 4.3.

4.1.1 Traffic Network Setup

We model a hypothetical traffic network using SUMO (See Figure 4.1). This traffic network was modeled in the form of a grid, with 4 vertical routes intercepting 5 horizontal routes, resulting to 25 junctions. Each route has 4 links and all the junctions are controlled. The traffic routes are detailed as follow:

- two routes from west to east. One route has links with two lanes and the other route has links with four lanes.
- two routes from east to west. One route has links with two lanes and the other route has links with four lanes.
- two routes from south to north with links of two lanes.
- two routes from north to south with links of two lanes.
- one bi-directional vertical route with two lanes in each direction.

Vehicles start to run in the traffic network each 20 seconds in the bi-directional route and each 20 seconds in the others routes. Vehicles can run through the routes with a maximum speed of 16,66 m/s. We used route blocks of 41 meters.

4.1.2 Emergency Incident Scenario

We simulate an emergency incident occurring at a point E of the urban traffic network, as shown in Figure 4.1. This incident

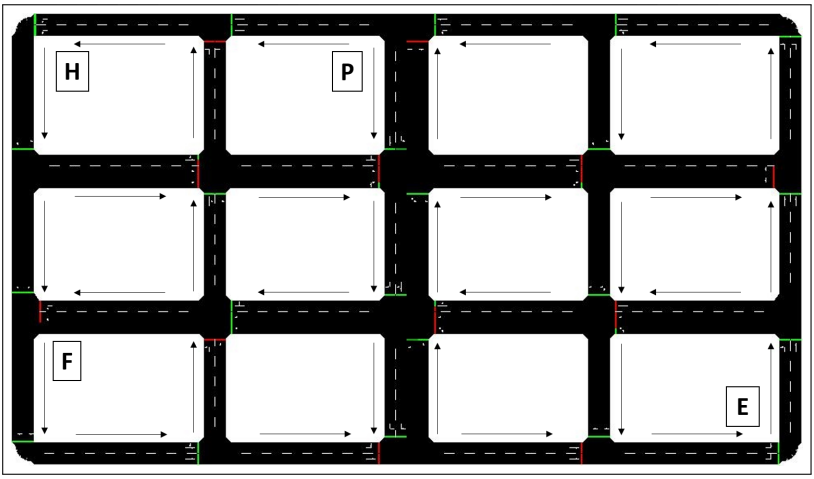

Figure 4.1: The hypothetical traffic network.

involves m victims that should be evacuated to the health care location H. The police is located at point P, the fire fighters are located at point F and ambulances are located at the health care location H.

Following, we list some of the activities usually performed during emergency response processes as actions that should need to be accomplished:

1. The police should go to E in order to ensure the security and order;
2. Fire-fighters should go to E in order to stop the fire;
3. Ambulances should go to E in order to evacuate all the victims to H;
4. The responders should return back to their base after finishing their tasks.
5. The junctions have to assign priority to responders.

Notice that we consider only hypothetical data in the setup, since the objective of this assessment is to evaluate the effectiveness of our proposal in relation to some scenarios introduced in the sequence, given a traffic network. Future works will consider realism in the assessment.

4.1.3 Assessment Metrics

To assess the first objective, we measure the time needed to perform all the emergency response process. We call this metric *emergency total time*. The objective of such metric is to assess how effectively our proposal reduces the responders travel time, which results into the reduction of the emergency response time.

The second objective is assessed using the following traffic metrics:

- *Average speed*: represents the average speed of all vehicles during the emergency response process.
- *Average travel time*: represents the average travel time of all vehicles during the emergency response process.
- *Average density*: represents the average number of vehicles by kilometers on the traffic links during the emergency response process.

We compared the following control scenarios:

- *Conventional control*: is the scenario in which the traffic is controlled by a fixed time traffic control strategy.
- *MAS distance control*: refers to the traffic and responders controlled by the proposed MAS and strategy, in which the responders best path is determined using the distance as link weight.
- *MAS density control*: refers to the traffic and responders controlled by the proposed MAS and strategy, in which the responders best path is determined using the density as link weight.

Being the effectiveness the degree to which something is successful in producing a desired result (Oxford 2015), for each metric, we define a desired value considered as reference in order to assess the effectiveness of our proposal in relation to the conventional control.

For the emergency total time, we consider as reference the total time in which the responders travel without the presence of other vehicles, in order to maximize their speed. In the case of the other metrics, the values obtained from the conventional control scenario will serve as reference, since the second objective of our proposal is to minimize the impact of priorities assigned to responders on the traffic flow.

This assessment aims to evaluate the effectiveness of our proposal to reach the two objectives established in Section 1.2, since these two objectives compose the aim of this dissertation. In this way, the traffic network setup and the emergency scenario introduced in Section 4.1 and 4.1.2 respectively, represent the simulation of an emergency that our proposal needs to solve.

We defined in the sequence, some metrics that helps to assess our proposal in relation to the two objectives established. The emergency total time assesses if our proposal was able to reduce the responders displacement comparing to a conventional control strategy, based on a reference scenario described above. The others metrics assess if our proposal reduced the impact of priorities assigned to responders on the traffic flow.

4.2 DETERMINATION OF α

As introduced in Section 3.2.2.2, α represents the number of tlc agents that prioritize at the same time a responder r at junctions j on a path p; $\alpha \in \{1, n\}$, n corresponds to the number of j on p. To determine α, the MAS distance control and MAS density control were assessed, and the results shown in the sequence were found after various simulations using $\alpha \in [1, 7]$.

Note that the highest number of junctions on a single path in the simulated traffic system was seven. We simulated emergencies with 50 victims, 4 ambulances and fire-fighters and 5 police vehicles. We use the emergency total time, the average speed and density as parameters in order to determine α. Figures 4.2, 4.3 and 4.4 show simulation results for the definition of α.

Figure 4.2 shows that the emergency total time is inversely proportional with the value of α. If we considered only this metric, we would use $\alpha = 7$ for our simulations.

We notice in Figure 4.3, that the average speed tends to decrease with the increase of α. This result shows how anticipating the responders priority tends to affect the traffic state.

In Figure 4.4, the density increase with the increase of α. Similarly, with the average speed, this metric also shows that the traffic state is affected with the increase of α. If we considered only the average speed and density metrics, we would use $\alpha = 1$ for our simulations.

With these results, we observe that the value of α depends of the objective of the traffic control. We opted to use $\alpha = 1$, since

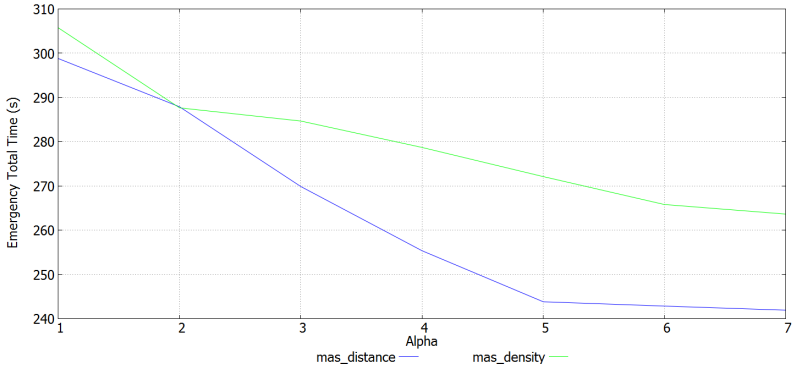

Figure 4.2: The emergency total time.

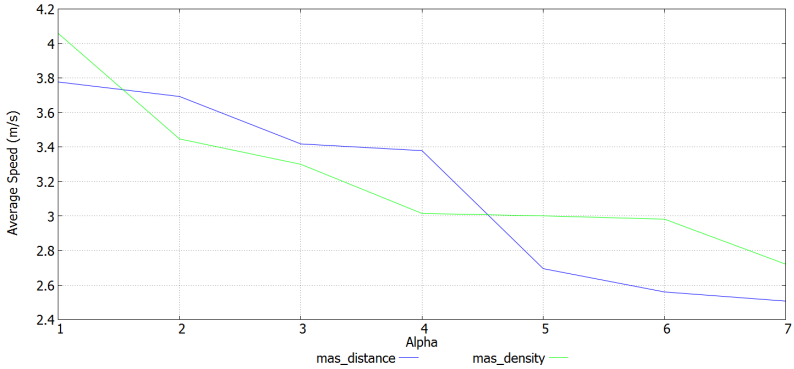

Figure 4.3: The average speed.

in this proposal, we give more emphasis to the second objective in order to prevent that the affected traffic flow be a perturbation of future emergencies that could occurs at the same urban traffic area.

4.3 ASSESSMENT RESULTS

We compare the scenarios described in Section 4.1.3 simulating emergencies with different number of victims, from 20 to 100 victims. We use the same number of responders for all the emergencies, that is: four ambulances and fire-fighters, and five police vehicles.

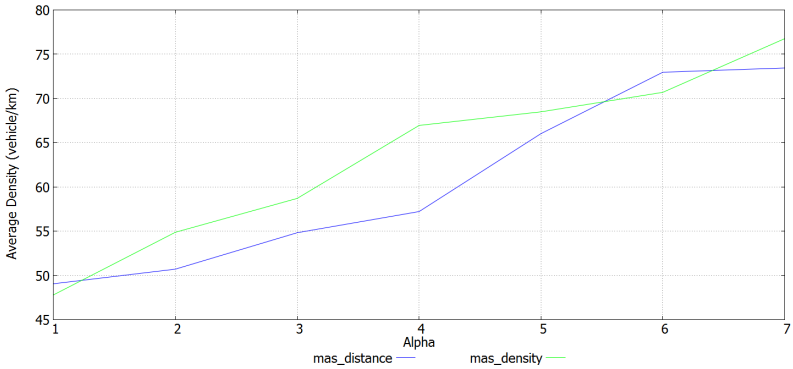

Figure 4.4: The average density.

4.3.1 First Objective Assessment

The emergency total time metric indicated the effectiveness of our proposal in reducing the responders travel time. Figure 4.5 shows the emergency total time relatively to the number of victims.

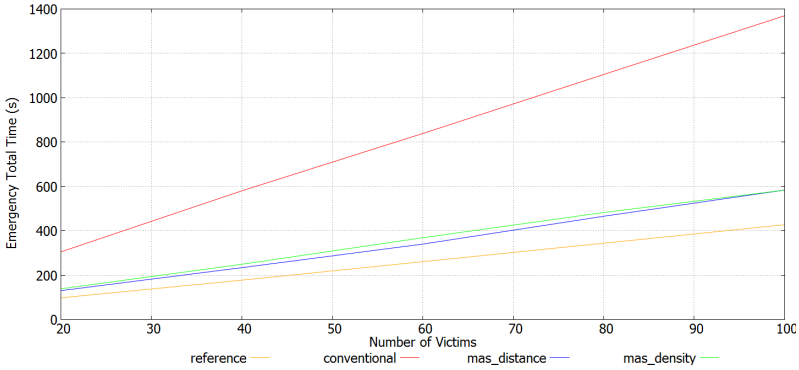

Figure 4.5: The emergency total time.

We calculated the emergency total time average for each of the scenarios in order to compare them numerically. The results are presented below.

- *Conventional*: $average = 1,016.9s$;
- *MAS density*: $average = 439.6s$;

- *MAS distance*: $average = 426.8s$;
- *Reference*: $average = 316.8s$;

As expected, the reference implied to the lowest emergency total time for all the number of victims. The time obtained by MAS distance and MAS density are close and significantly below than the time from conventional scenario.

Comparing these emergency total time averages, we conclude that the proposed MAS with the control strategy reduced meaningfully the emergency total time comparing to the conventional control strategy. Therefore, our proposal was effective in improving the responders displacement to their destinations during the emergency incident simulated.

4.3.2 Second Objective Assessment

The average speed, travel time and density indicated whether the traffic state was affected by the control strategy used to route the responders during the emergency response process. Figures 4.6, 4.7 and 4.8 show the results respectively.

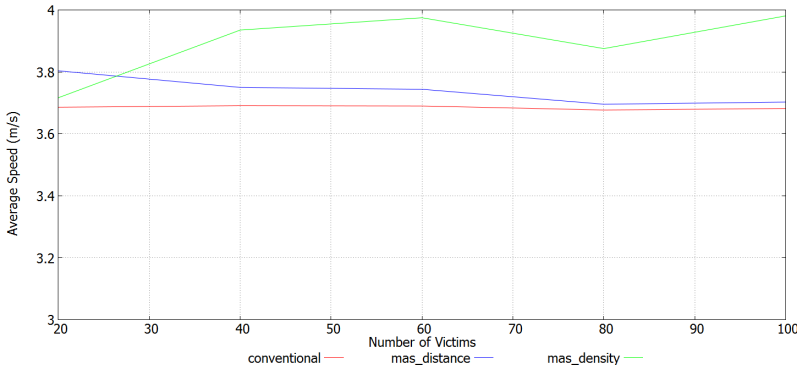

Figure 4.6: The average speed.

According to the graphic depicted in Figure 4.6, the average speeds relative to MAS distance and MAS density are almost the same compared to Conventional, which is also the reference in this case. Therefore, the proposed MAS model and control strategy have not affected the average speed of vehicles into the traffic. This also occurs with the travel time and density.

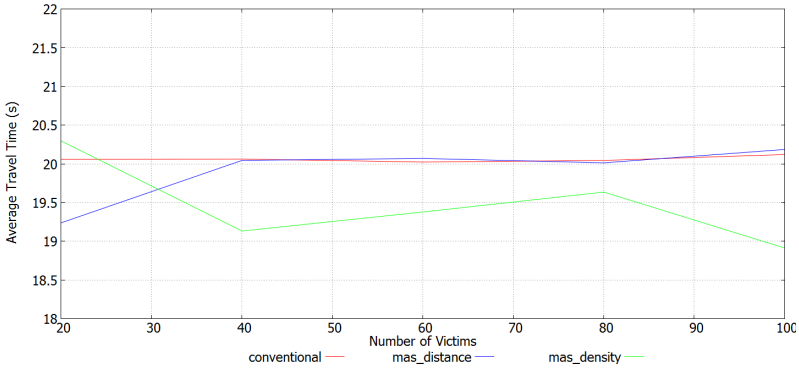

Figure 4.7: The average travel time.

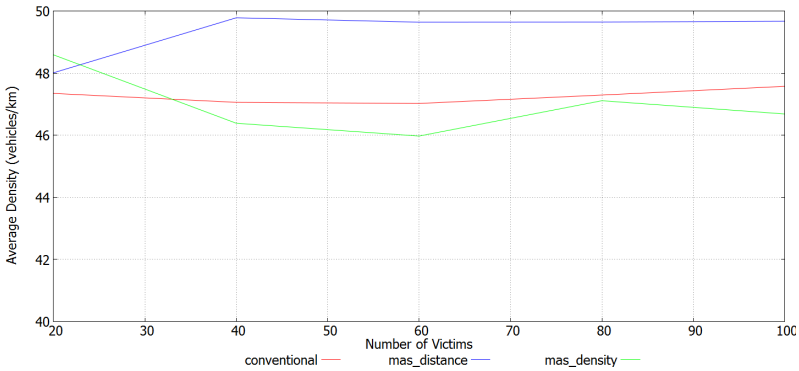

Figure 4.8: The average density.

4.3.3 Use of MAS Approach

We do not assess quantitatively the use of MAS approach, we highlight below some benefits observed in using the MAS approach in our proposal.

4.3.3.1 Programming Paradigm

The Agent Oriented Programming (AOP) Paradigm and the *Jacamo* framework were helpful in the implementation of our proposal, providing powerful features to deal with issues such as concurrency. Agents plans are triggered concurrently according to their beliefs or perceptions in the environment, in order to achieve their

goals. The exchange of messages between the agents is also concurrent and can be asynchronous or synchronous. Our simulations, for example, involved one a_{anl} interacting with the environment and with 13 a_{rtr} agents, which were also interacting with the environment and with 25 a_{tlc} agents. All these interactions were concurrent. We did not need to implement explicitly such concurrency, since the concurrency is part of the *AOP* paradigm.

4.3.3.2 Cooperation

The use of MAS approach facilitated the cooperation during the emergency response process on two main elements.

- How to approach the problem: using the task sharing cooperation technique, we were able to decompose the overall problem into several small problems then to build the goals tree, following the AOSE methodology, and assigning group of goals to different agents. This provided us the capacity to know clearly in which circumstances and for which issues agents would need the information or actions of others agents. That is, when, where and how to cooperate.

- The interactions: that is the key of the cooperation. Knowing, when, where and how to cooperate, the cooperation among the agents was implemented by the interactions following the interaction protocols defined. Using the *JaCaMo* framework, such protocols are traduced to messages sent among the agents. The implementation of such messages is very simple and, as said before, all the concurrency is implicit.

In sum, using MAS we could know when, where and how the cooperation would occur: decomposing the overall problem into small problems, establishing goals for each small problem, and assigning to agents different responsibilities according to the goal tree. In this way, the cooperation was implemented through the interactions among the agents following some protocols defined. Notice that all of these concepts are part of MAS approach and the *Jacamo* framework provides features for their implementation. In traditional distributed systems, these should be modeled, implemented and treated explicitly, since these system almost provide only the communication means between sub systems.

4.3.3.3 Coordination

The use of MAS approach, specially the organization, facilitated the coordination of cooperative activities between the agents. With the organization configuration, the roles specified in the structural dimension committed agents to missions specified in the functional dimension involving partials goals in order to achieve the overall goal. The commitment of the agents to missions followed norms specified in the normative dimension, serving as a convention to monitor them.

Moreover, using *JaCaMo*, agents plans were triggered by some events. In the interaction protocols, each action to be performed by an agent was preceded by an event, which corresponded generally to a message received from another agent or a perception from the environment. This behavior made agents plans to be coordinated following the interaction protocols. In this way, the coordination was implicit, in agents plans, and explicit, defined by the interaction protocols.

All these concepts and specifications provided the coordination of agents activities and are part of MAS approach using *JaCaMo* framework. In traditional systems, all these concepts should be modeled and implemented explicitly.

4.4 ANALYSIS OF THE PROPOSAL

We use this section to analyze our proposal, considering all the observations highlighted in Section 2.4.3 in Chapter **??**. We noticed that some works in the literature focus more on emergency management at and around the local of the incident than on traffic control. Our proposal consider and focus on the routing of emergency vehicles with the objective of improving their travel time and minimizing the impact of priorities given to them on the traffic.

Moreover, we also noticed works which focus on traffic control in emergency situations using MAS approaches. However, most of these works does not present a clear cooperation technique or/and coordination mechanism. There are also works in which the cooperation technique is quite mixed or sometimes confused with the coordination in MAS. Our proposal use the task sharing cooperation technique and the coordination through joint intention. Both of the techniques are presented in Section 3.3.5 and 3.3.6 respectively.

Finally, there are works that consider almost everything as agents in their representation of the MAS. Some others works even

use the environment approach, but they still mix and/or confuse the endogenous environment with the exogenous environment. Our proposal considers the traffic system as an exogenous environment controlled by agents in the MAS using an endogenous environment. The MAS with all its levels is introduced in Section 2.1.

5 CONCLUSION

5.1 CONCLUSION

We studied, in this dissertation, the problem of improving an emergency response process in an urban traffic. We approached this problem proposing a Multi Agent System endowed with an explicit cooperation technique, which has implemented a strategy that control the traffic signals and route emergency vehicles in order to achieve the two objectives established in Chapter 1, Section 1. We defined the conventional control, the MAS distance control and the MAS density control, as scenarios to be compared for the assessment. We also defined the emergency total time as metric for the first objective, as well as, the average speed, travel time and density, as metrics for the second objective. Finally, for each of the metrics, we defined a reference value.

The assessment results showed that our proposal was able to reduce the travel time of emergency vehicles, since the emergency total time obtained using our proposal was significantly less than the total time obtained using the conventional control scenario, considering the reference value found. Furthermore, our proposal was able to minimize the impact of priorities given to emergency vehicles on the traffic flow, since the average speed, travel time and density obtained with our proposal were almost the same in relation to the values obtained using the conventional control scenario.

Nevertheless, one of the limitations in our proposal is the use of isolated traffic control strategy, in which each junction is managed independently of the others, seems to affect seriously the traffic flow performance in situations where junctions are close. As commented in Section 3.2.3.1, coordination in our traffic control strategy is left as future work.

Furthermore, our assessment does not present deeply realistic data, such as the traffic network and traffic flow of a small city; and the conventional control strategy used is not optimal. However this dissertation did not focused on improving the traffic control or assessing the proposal based on an optimized traffic lights control, future works will consider such characteristics.

Moreover, we do not compare our proposal to another from the literature in the assessment. However in Chapter 2 Section 2.4, we highlight our contribution in relation to works presented therein, a deep assessment of those proposals comparing to our seems to be important. Future works will present such assessments.

Finally, as future works, we also intend to consider in our

proposal, the use of more than one a_{anl} in order to control several emergencies occurring simultaneously at a same urban area. Questions like: how resources such as ambulances will be distributed in order to save a maximum number of lives; how responders will be managed, etc., surge when consider such scenario.

BIBLIOGRAPHY

Au et al. 2015 AU, T.-C.; ZHANG, S.; STONE, P. Autonomous intersection management for semi-autonomous vehicles. In: *Handbook of Transportation*. [s.n.], 2015. Available from Internet: <http://www.cs.utexas.edu/users/ai-lab/?au:hot15>.

Bernon et al. 2005 BERNON, C.; COSSENTINO, M.; PAVÓN, J. Agent-oriented software engineering. *The Knowledge Engineering Review*, Cambridge Univ Press, v. 20, n. 02, p. 99–116, 2005.

Borges et al. 2010 BORGES, M. R. S. et al. Assigning emergency vehicles to urban incidents. In: *Frontiers in Artificial Intelligence and Applications*. [S.l.: s.n.], 2010. v. 212, p. 498–509. ISBN 9781607505761. ISSN 09226389.

Burmeister et al. 1997 BURMEISTER, B.; HADDADI, A.; MATYLIS, G. Application of multi-agent systems in traffic and transportation. In: IET. *Software Engineering. IEE Proceedings-[see also Software, IEE Proceedings]*. [S.l.], 1997. v. 144, n. 1, p. 51–60.

Cetin and Jordan 2012 CETIN, M.; JORDAN, C. A. Making way for emergency vehicles at oversaturated signals under vehicle-to-vehicle communications. In: *Vehicular Electronics and Safety (ICVES), 2012 IEEE International Conference on*. [S.l.: s.n.], 2012. p. 279–284.

Changhong et al. 2002 CHANGHONG, L.; MINQIANG, L.; JISONG, K. Cooperation structure of multi-agent and algorithms. In: IEEE. *Artificial Intelligence Systems, 2002.(ICAIS 2002). 2002 IEEE International Conference on*. [S.l.], 2002. p. 303–307.

Cormen 2009 CORMEN, T. H. *Introduction to algorithms*. [S.l.]: MIT press, 2009.

Demazeau 1995 DEMAZEAU, Y. From interactions to collective behaviour in agent-based systems. In: CITESEER. *In: Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo*. [S.l.], 1995.

DENATRAN 2014 DENATRAN. 2014. <http://www.denatran.gov.br/download/resolucoes/resolucao4832014_anexo.pdf>. Accessed at 26/11/2015.

Dillon 2014 DILLON, B. *Blackstone's Emergency Planning, Crisis, and Disaster Management*. [S.l.]: Oxford University Press, 2014.

Djahel et al. 2015 DJAHEL, S. et al. Reducing emergency services response time in smart cities: An advanced adaptive and fuzzy approach. In: *Smart Cities Conference (ISC2), 2015 IEEE First International*. [S.l.: s.n.], 2015. p. 1–8.

Dresner and Stone 2008 DRESNER, K.; STONE, P. A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research*, p. 591–656, 2008.

Durfee 2001 DURFEE, E. H. Distributed problem solving and planning. In: *Multi-agent systems and applications*. [S.l.]: Springer, 2001. p. 118–149.

Fagel 2010 FAGEL, M. J. *Principles of emergency management and emergency operations centers (EOC)*. [S.l.]: CRC press, 2010.

Ferber et al. 2004 FERBER, J.; GUTKNECHT, O.; MICHEL, F. From agents to organizations: An organizational view of multi-agent systems. In: *Agent-Oriented Software Engineering IV*. [S.l.]: Springer, 2004. p. 214–230.

FIPA 2002 FIPA. 2002. <http://www.fipa.org/index.html>. Accessed at 09/04/2015.

FIPA Agent Communication Language Specifications 2002 FIPA Agent Communication Language Specifications. 2002. <http://www.fipa.org/repository/aclspecs.html>. Accessed at 09/04/2015.

FIPA Contract Net Interaction Protocol Specification 2002 FIPA Contract Net Interaction Protocol Specification. 2002. <http://www.fipa.org/specs/fipa00029/SC00029H.html>. Accessed at 09/04/2015.

FIPA Request Interaction Protocol Specification 2002 FIPA Request Interaction Protocol Specification. 2002. <http://www.fipa.org/specs/fipa00026/SC00026H.html>. Accessed at 09/04/2015.

Gasser 1992 GASSER, L. An overview of dai. *Distributed Artificial Intelligence: Theory and Praxis*, v. 9, n. 9-29, p. 28, 1992.

Gomez-Sanz and Pavon 2006 GOMEZ-SANZ, J. J.; PAVON, J. Defining coordination in multi-agent systems within an agent oriented software engineering methodology. In: ACM. *Proceedings of the 2006 ACM symposium on Applied computing*. [S.l.], 2006. p. 424–428.

Guberinic et al. 2007 GUBERINIC, S.; SENBORN, G.; LAZIC, B. *Optimal traffic control: urban intersections*. [S.l.]: CRC Press, 2007.

Hubner et al. 2007 HUBNER, J. F.; SICHMAN, J. S.; BOISSIER, O. Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, Inderscience Publishers, v. 1, n. 3-4, p. 370–395, 2007.

Hubner Rafael Heitor Bordini 2015 HUBNER RAFAEL HEITOR BORDINI, O. B. A. R. A. S. J. F. *The JaCaMo approach*. 2015. <http://jacamo.sourceforge.net/?page_id=2>.

Jeng et al. 2013 JENG, A. A. K. et al. Adaptive urban traffic signal control system with bus priority. In: *Vehicular Technology Conference (VTC Spring), 2013 IEEE 77th*. [S.l.: s.n.], 2013. p. 1–5. ISSN 1550-2252.

Jennings 1999 JENNINGS, N. R. Agent-oriented software engineering. In: *Multiple Approaches to Intelligent Systems*. [S.l.]: Springer, 1999. p. 4–10.

Jennings 2000 JENNINGS, N. R. On agent-based software engineering. *Artificial intelligence*, Elsevier, v. 117, n. 2, p. 277–296, 2000.

Kang et al. 2013 KANG, W. et al. A heuristic implementation of emergency traffic evacuation in urban areas. In: *Proceedings of 2013 IEEE International Conference on Service Operations and Logistics, and Informatics*. IEEE, 2013. p. 40–44. ISBN 978-1-4799-0530-0. Available from Internet: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6611378>.

Kartalopoulos 1999 KARTALOPOULOS, S. V. Congestion control. In: ____. *Understanding SONET/SDH and ATM:Communications Networks for the Next Mellennium*. Wiley-IEEE Press, 1999. p. 181–184. ISBN 9780470546857. Available from Internet: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5270902>.

Kosmatopoulos et al. 2007 KOSMATOPOULOS, E. B. et al. Adaptive fine-tuning of nonlinear control systems with application to the urban traffic control strategy tuc. *IEEE Transactions on Control Systems Technology*, v. 15, n. 6, p. 991–1002, Nov 2007. ISSN 1063-6536.

Krajzewicz et al. 2012 KRAJZEWICZ, D. et al. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, v. 5, n. 3&4, p. 128–138, December 2012.

Li et al. 1996 LI, M. et al. A cooperative intelligent system for urban traffic problems. In: IEEE. *Intelligent Control, 1996., Proceedings of the 1996 IEEE International Symposium on*. [S.l.], 1996. p. 162–167.

Liu et al. 2007 LIU, H. X. et al. *Model Reference Adaptive Control Framework for Real-Time Traffic Management under Emergency Evacuation*. 2007. 43–50 p.

Lu 2009 LU, F. Rapid Handling of Urban Traffic Emergencies Based on Decision and Command Support System. *2009 Second International Conference on Intelligent Computation Technology and Automation*, Ieee, p. 677–680, 2009. Available from Internet: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5288073>.

Luck et al. 2005 LUCK, M. et al. Agent technology: computing as interaction (a roadmap for agent based computing). In: . [S.l.]: University of Southampton, 2005.

Mala 2012 MALA, M. A multi-agent system based approach to emergency management. In: *2012 6th IEEE International Conference Intelligent Systems*. [S.l.: s.n.], 2012. p. 095–101. ISSN 1541-1672.

Malone and Crowston 1994 MALONE, T. W.; CROWSTON, K. The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)*, ACM, v. 26, n. 1, p. 87–119, 1994.

Malowanczyk 2014 MALOWANCZYK, T. K. Interaction in multi-agent systems. In: . [S.l.]: Technical University of Denmark, Department of Applied Mathematics and Computer Science, 2014.

McCluskey et al. 2016 MCCLUSKEY, T. L. et al. *Autonomic Road Transport Support Systems*. [S.l.]: Springer, 2016.

McCoy et al. 2013 MCCOY, C. E. et al. Emergency medical services out-of-hospital scene and transport times and their association with mortality in trauma patients presenting to an urban level i trauma center. *Annals of emergency medicine*, Elsevier, v. 61, n. 2, p. 167–174, 2013.

Min and Jin 2013 MIN, G.; JIN, X. Analytical modelling and optimization of congestion control for prioritized multi-class self-similar traffic. *IEEE Transactions on Communications*, v. 61, n. 1, p. 257–265, January 2013. ISSN 0090-6778.

Mohan 2006 MOHAN, D. *Road traffic injury prevention training manual*. [S.l.]: World Health Organization, 2006.

Monares et al. 2012 MONARES, A. et al. Improving the initial response process in urban emergencies. In: *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2012*. [S.l.: s.n.], 2012. p. 379–386. ISBN 9781467312127.

Noori 2013 NOORI, H. Modeling the impact of vanet-enabled traffic lights control on the response time of emergency vehicles in realistic large-scale urban area. In: *2013 IEEE International Conference on Communications Workshops (ICC)*. [S.l.: s.n.], 2013. p. 526–531. ISSN 2164-7038.

Oxford 2015 OXFORD. *Oxford dictionaries*. 2015. Http://www.oxforddictionaries.com/pt/definiAcessed at November/2015.

Patrascu et al. 2015 PATRASCU, M.; ION, A.; CONSTANTINESCU, V. Agent based simulation applied to the design of control systems for emergency vehicles access. In: *ITS Telecommunications (ITST), 2015 14th International Conference on*. [S.l.: s.n.], 2015. p. 50–54.

Prometheus AEOlus MEthod 2013 PROMETHEUS AEOlus MEthod. 2013. <http://user.das.ufsc.br/~jomi/das6607/metodologia-daniela/aeolus_31102013.pdf>. Accessed at 07/2015.

Ricci et al. 2011 RICCI, A.; PIUNTI, M.; VIROLI, M. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, Springer, v. 23, n. 2, p. 158–192, 2011.

Riedel and Brunner 1994 RIEDEL, T.; BRUNNER, U. Traffic control using graph theory. *Control Engineering Practice*, Elsevier, v. 2, n. 3, p. 397–404, 1994.

Rodríguez et al. 2009 RODRÍGUEZ, A. et al. A multi-agent system for traffic control for emergencies by quadrants. In: IEEE. *Internet and Web Applications and Services, 2009. ICIW'09. Fourth International Conference on*. [S.l.], 2009. p. 247–253.

SAMU 2015 SAMU, S. *Relatório Estatístico Ano 2015*. [S.l.]: Associação Paulista para o Desenvolvimento da Medicina -SPDM, Serviço de Atendimento Móvel de (SAMU), 2015.

Schumann 2012 SCHUMANN, R. Engineering coordination: Selection of coordination mechanisms. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 7068 LNAI, p. 164–186, 2012.

Shumin et al. 2010 SHUMIN, S.; ZHAOSHENG, Y.; MAOLEI, Z. A decision support system of urban traffic emergency control based on expert system. In: *2010 IEEE International Conference on Software Engineering and Service Sciences*. [s.n.], 2010. p. 221–225. ISBN 978-1-4244-6054-0. Available from Internet: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5552415>.

Stefanello et al. 2013 STEFANELLO, F. et al. On the minimization of traffic congestion in road networks with tolls. *Annals of Operations Research*, Springer, p. 1–21, 2013.

Sundar et al. 2015 SUNDAR, R.; HEBBAR, S.; GOLLA, V. Implementing intelligent traffic control system for congestion control, ambulance clearance, and stolen vehicle detection. *IEEE Sensors Journal*, v. 15, n. 2, p. 1109–1113, Feb 2015. ISSN 1530-437X.

Tomás and Garcia 2005 TOMÁS, V. R.; GARCIA, L. A. A cooperative multiagent system for traffic management and control. In: ACM. *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. [S.l.], 2005. p. 52–59.

Trent et al. 2008 TRENT, S. et al. *Designing to Support Command and Control in Urban Firefighting*. [S.l.], 2008.

Vilarinho et al. 2015 VILARINHO, C.; TAVARES, J. P.; ROSSETTI, R. J. A conceptual mas model for real-time traffic control. In: *Progress in Artificial Intelligence*. [S.l.]: Springer, 2015. p. 157–168.

Viriyasitavat and Tonguz 2012 VIRIYASITAVAT, W.; TONGUZ, O. K. Priority management of emergency vehicles at intersections using self-organized traffic control. In: *Vehicular Technology Conference (VTC Fall), 2012 IEEE*. [S.l.: s.n.], 2012. p. 1–4. ISSN 1090-3038.

Weyns et al. 2007 WEYNS, D.; OMICINI, A.; ODELL, J. Environment as a first class abstraction in multiagent systems. *Autonomous agents and multi-agent systems*, Springer, v. 14, n. 1, p. 5–30, 2007.

WHO 2013 WHO. *Global Status Report on Road Safety 2013*. [S.l.]: World Health Organization, 2013.

WHO 2015 WHO. *Country Profiles*. [S.l.]: World Health Organization, 2015. <http://www.who.int/violence_injury_prevention/road_safety_status/2015/Country_profiles_combined_GSRRS2015_2.pdf?ua=1>.

WHO 2015 WHO. *Road Traffic Injuries 2015*. [S.l.]: World Health Organization, 2015. <http://www.who.int/mediacentre/factsheets/fs358/en/>.

Wooldridge 2009 WOOLDRIDGE, M. *An introduction to multiagent systems*. [S.l.]: John Wiley & Sons, 2009.

Wooldridge and Ciancarini 1999 WOOLDRIDGE, M.; CIANCARINI, P. Agent-oriented software engineering. *Dept of Computer Science, University of Liverpool and Dipartimento di Scienze dell'Informazione, University of Bologna. Handbook of Software Engineering and Knowledge Engineering Vol. 0, No. 0*, Citeseer, 1999.

Wooldridge et al. 2000 WOOLDRIDGE, M.; JENNINGS, N. R.; KINNY, D. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and multi-agent systems*, Springer, v. 3, n. 3, p. 285–312, 2000.

Wu et al. 2010 WU, Z. et al. Traffic organization method for emergency evacuation based on information centrality. In: *Proceedings - 2nd IEEE International Conference on Advanced Computer Control, ICACC 2010*. [S.l.: s.n.], 2010. v. 3, p. 92–96. ISBN 9781424458462.

Yoo et al. 2012 YOO, C. H. et al. Hardware-in-the-loop simulation of dc microgrid with multi-agent system for emergency demand response. In: *2012 IEEE Power and Energy Society General Meeting*. [S.l.: s.n.], 2012. p. 1–6. ISSN 1932-5517.

Zeng et al. 2014 ZENG, X. et al. A real-time transit signal priority control model considering stochastic bus arrival time. *IEEE Transactions on Intelligent Transportation Systems*, v. 15, n. 4, p. 1657–1666, Aug 2014. ISSN 1524-9050.

IMPLEMENTATION DETAILS

We present in this appendix details about the implementation. That is, SUMO configurations in *XML* formats, agents implementation in *Jason*, and Java class implementing the MAS environment artifacts and the traffic system. The implementation content is divided in: traffic system, MAS and Network interface.

URBAN TRAFFIC SYSTEM IMPLEMENTATION

We present in the sequence the XML content and Java class representing SUMO configurations of the traffic network elements and the traffic system implementation using *TraCI4J*.

```
<?xml version="1.0" encoding="UTF-8"?>
<nodes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/nodes_file.xsd">
    <node id="J0" x="0.0" y="0.0" type="traffic_light"/>
    <node id="J1" x="0.0" y="23.5" type="traffic_light"/>
    <node id="J2" x="0.0" y="60.0" type="traffic_light"/>
    <node id="J3" x="0.0" y="96.5" type="traffic_light"/>
    <node id="J4" x="0.0" y="120.0" type="traffic_light"/>

    <node id="J5" x="50.0" y="0.0" type="traffic_light"/>
    <node id="J6" x="50.0" y="23.5" type="traffic_light"/>
    <node id="J7" x="50.0" y="60.0" type="traffic_light"/>
    <node id="J8" x="50.0" y="96.5" type="traffic_light"/>
    <node id="J9" x="50.0" y="120.0" type="traffic_light"/>

    <node id="J10" x="110.0" y="0.0" type="traffic_light"/>
    <node id="J11" x="110.0" y="23.5" type="traffic_light"/>
    <node id="J12" x="110.0" y="60.0" type="traffic_light"/>
    <node id="J13" x="110.0" y="96.5" type="traffic_light"/>
    <node id="J14" x="110.0" y="120.0" type="traffic_light"/>

    <node id="J15" x="170.0" y="0.0" type="traffic_light"/>
    <node id="J16" x="170.0" y="23.5" type="traffic_light"/>
    <node id="J17" x="170.0" y="60.0" type="traffic_light"/>
    <node id="J18" x="170.0" y="96.5" type="traffic_light"/>
    <node id="J19" x="170.0" y="120.0" type="traffic_light"/>

    <node id="J20" x="220.0" y="0.0" type="traffic_light"/>
    <node id="J21" x="220.0" y="23.5" type="traffic_light"/>
    <node id="J22" x="220.0" y="60.0" type="traffic_light"/>
    <node id="J23" x="220.0" y="96.5" type="traffic_light"/>
    <node id="J24" x="220.0" y="120.0" type="traffic_light"/>
</nodes>
```

Figure 1: SUMO nodes.nod.xml.

MAS IMPLEMENTATION

We present in the sequence the Jason file contents, the Java classes and the XML content representing MAS Agents, environment

```
<?xml version="1.0" encoding="UTF-8"?>
<edges xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/edges_file.xsd">
    <!-- Vertical links -->
    <edge id="J1J0" from="J1" to="J0" type="s"/>
    <edge id="J2J1" from="J2" to="J1" type="s"/>
    <edge id="J3J2" from="J3" to="J2" type="s"/>
    <edge id="J4J3" from="J4" to="J3" type="s"/>

    <edge id="J5J6" from="J5" to="J6" type="s"/>
    <edge id="J6J7" from="J6" to="J7" type="s"/>
    <edge id="J7J8" from="J7" to="J8" type="s"/>
    <edge id="J8J9" from="J8" to="J9" type="s"/>

    <edge id="J10J11" from="J10" to="J11" type="s"/>
    <edge id="J11J12" from="J11" to="J12" type="s"/>
    <edge id="J12J13" from="J12" to="J13" type="s"/>
    <edge id="J13J14" from="J13" to="J14" type="s"/>
    <edge id="J11J10" from="J11" to="J10" type="s"/>
    <edge id="J12J11" from="J12" to="J11" type="s"/>
    <edge id="J13J12" from="J13" to="J12" type="s"/>
    <edge id="J14J13" from="J14" to="J13" type="s"/>

    <edge id="J16J15" from="J16" to="J15" type="s"/>
    <edge id="J17J16" from="J17" to="J16" type="s"/>
    <edge id="J18J17" from="J18" to="J17" type="s"/>
    <edge id="J19J18" from="J19" to="J18" type="s"/>

    <edge id="J20J21" from="J20" to="J21" type="s"/>
    <edge id="J21J22" from="J21" to="J22" type="s"/>
    <edge id="J22J23" from="J22" to="J23" type="s"/>
    <edge id="J23J24" from="J23" to="J24" type="s"/>

    <!-- Horizontal links -->

    <edge id="J0J5" from="J0" to="J5" type="s"/>
    <edge id="J5J10" from="J5" to="J10" type="s"/>
    <edge id="J10J15" from="J10" to="J15" type="s"/>
    <edge id="J15J20" from="J15" to="J20" type="s"/>

    <edge id="J6J1" from="J6" to="J1" type="s"/>
    <edge id="J11J6" from="J11" to="J6" type="s"/>
    <edge id="J16J11" from="J16" to="J11" type="s"/>
    <edge id="J21J16" from="J21" to="J16" type="s"/>

    <edge id="J2J7" from="J2" to="J7" type="s"/>
    <edge id="J7J12" from="J7" to="J12" type="s"/>
    <edge id="J12J17" from="J12" to="J17" type="s"/>
    <edge id="J17J22" from="J17" to="J22" type="s"/>
    <edge id="J7J2" from="J7" to="J2" type="s"/>
    <edge id="J12J7" from="J12" to="J7" type="s"/>
    <edge id="J17J12" from="J17" to="J12" type="s"/>
    <edge id="J22J17" from="J22" to="J17" type="s"/>

    <edge id="J3J8" from="J3" to="J8" type="s"/>
    <edge id="J8J13" from="J8" to="J13" type="s"/>
    <edge id="J13J18" from="J13" to="J18" type="s"/>
    <edge id="J18J23" from="J18" to="J23" type="s"/>

    <edge id="J9J4" from="J9" to="J4" type="s"/>
    <edge id="J14J9" from="J14" to="J9" type="s"/>
    <edge id="J19J14" from="J19" to="J14" type="s"/>
    <edge id="J24J19" from="J24" to="J19" type="s"/>
</edges>
```

Figure 2: SUMO edges.edg.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<types>
    <type id="p" priority="1" numLanes="3" speed="16.66"/>
    <type id="s" priority="1" numLanes="2" speed="16.66"/>
    <type id="t" priority="1" numLanes="1" speed="13.88"/>
</types>
```

Figure 3: SUMO types.typ.xml.

```
<configuration>
    <input>
        <edge-files value="edges.edg.xml"/>
        <node-files value="nodes.nod.xml"/>
        <type-files value="types.typ.xml"/>
    </input>
    <output>
        <output-file value="network.net.xml"/>
    </output>
    <processing>
        <no-turnarounds value="true"/>
    </processing>
</configuration>
```

Figure 4: SUMO network.cfg.xml.

```
<viewsettings>
    <scheme name="real world"/>
    <viewport y="60" x="110" zoom="100"/>
    <delay value="200"/>
</viewsettings>
```

Figure 5: SUMO settings.settings.xml.

artifacts and organization implementation respectively.

```
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/routes_file.xsd">
    <vTypeDistribution id="DEFAULT_VEHTYPE">
         <vType length="5.0" accel="3.0" decel="6.0" maxSpeed="22.2" sigma="0.5" id="passen
            guiShape="passenger" osgFile="vw_golf.3ds" probability="13" />
        <vType length="12.5" accel="2.0" decel="6.0" maxSpeed="16.7" sigma="0.5" id="bus" m
            guiShape="bus" osgFile="tour_bus.3ds" width="2.4" probability="3"/>
        <vType length="16.2" accel="1.0" decel="5.0" maxSpeed="11.1" sigma="0.5" id="transp
            minGap="2.5" guiShape="truck/trailer" osgFile="lorry.3ds" width="2.6" probabili

        <vType id="amb" accel="22.0" decel="0.5" sigma="1.0" length="5"
            maxSpeed="70" color="white" probability="0" />
        <vType id="plc" accel="22.0" decel="0.5" sigma="1.0" length="5"
            maxSpeed="70" color="blue" probability="0" />
        <vType id="frt" accel="22.0" decel="0.5" sigma="1.0" length="5"
            maxSpeed="70" color="red" probability="0" />
    </vTypeDistribution>
  <!-- Vertical Routes -->
  <route id="J4J0" edges="J4J3 J3J2 J2J1 J1J0"/>
  <flow id="F_J4J0" route="J4J0" begin="0" end="9000" period="10"/>

  <route id="J5J9" edges="J5J6 J6J7 J7J8 J8J9"/>
  <flow id="F_J5J9" route="J5J9" begin="0" end="9000" period="10"/>

  <route id="J14J10" edges="J14J13 J13J12 J12J11 J11J10"/>
  <flow id="F_J14J10" route="J14J10" begin="0" end="9000" period="10"/>
  <route id="J10J14" edges="J10J11 J11J12 J12J13 J13J14"/>
  <flow id="F_J10J14" route="J10J14" begin="0" end="9000" period="10"/>

  <route id="J19J15" edges="J19J18 J18J17 J17J16 J16J15"/>
  <flow id="F_J19J15" route="J19J15" begin="0" end="9000" period="10"/>

  <route id="J20J24" edges="J20J21 J21J22 J22J23 J23J24"/>
  <flow id="F_J20J24" route="J20J24" begin="0" end="9000" period="10"/>
  <!-- Horizontal Routes -->
  <route id="J0J20" edges="J0J5 J5J10 J10J15 J15J20"/>
  <flow id="F_J0J20" route="J0J20" begin="0" end="9000" period="10"/>

  <route id="J21J1" edges="J21J16 J16J11 J11J6 J6J1"/>
  <flow id="F_J21J1" route="J21J1" begin="0" end="9000" period="10"/>

  <route id="J2J22" edges="J2J7 J7J12 J12J17 J17J22"/>
  <flow id="F_J2J22" route="J2J22" begin="0" end="9000" period="10"/>
  <route id="J22J2" edges="J22J17 J17J12 J12J7 J7J2"/>
  <flow id="F_J22J2" route="J22J2" begin="0" end="9000" period="10"/>

  <route id="J3J23" edges="J3J8 J8J13 J13J18 J18J23"/>
  <flow id="F_J3J23" route="J3J23" begin="0" end="9000" period="10"/>

  <route id="J24J4" edges="J24J19 J19J14 J14J9 J9J4"/>
  <flow id="F_J24J4" route="J24J4" begin="0" end="9000" period="10"/>
  <!-- Mixes Routes -->
</routes>
```

Figure 6: SUMO routes.rou.xml.

NETWORK INTERFACE IMPLEMENTATION

We present in the sequence the Java class representing the network interface implementation.

```xml
<configuration>
    <input>
        <edge-files value="edges.edg.xml"/>
        <node-files value="nodes.nod.xml"/>
        <type-files value="types.typ.xml"/>
    </input>
    <output>
        <output-file value="network.net.xml"/>
    </output>
    <processing>
        <no-turnarounds value="true"/>
    </processing>
</configuration>
```

Figure 7: SUMO traffic.sumo.cfg.

```java
package world;

import java.awt.geom.Point2D;

/**
 * @author Emmanuel Katende Dinanga;
 *
 */
public class Traffic extends UnicastRemoteObject implements RemoteInterface {

    private static final long serialVersionUID = 1L;

    private static final String RMI_PATH = "rmi://127.0.0.1:2020/Traffic";
    private static final String TRAFFIC_CFG_FILE = "./src/env/sumo_files/traffic.sumo.cfg";
    private static final String SUMO_EXE_PROPERTY = "it.polito.appeal.traci.sumo_exe";
    private static final String SUMO_GUI_PATH = "C:/sumo-0.23.0/bin/sumo-gui.exe";

    private TrafficSim sim;
    private String emgLocal;
    private int numVictims;
```

Figure 8: Java Traffic.java.

```java
public Traffic(String emgLocal, int numVictims) throws IOException, InterruptedException {
    LocateRegistry.createRegistry(2020);
    Naming.rebind(RMI_PATH, this);
    sim = new TrafficSim(TRAFFIC_CFG_FILE, SUMO_GUI_PATH,
            SUMO_EXE_PROPERTY, 5100, true);
    this.emgLocal = emgLocal;
    this.numVictims = numVictims;
    sim.nextSimStep();
}

public static void main(String[] args) throws Exception {
    Traffic t = new Traffic("J20", 1);
}

public synchronized void runTraffic() {
    try {
        sim.nextSimStep();
    } catch (IllegalStateException | IOException e) {
        e.printStackTrace();
    }
}
```

Figure 9: Java Traffic.java.

```java
@Override
public synchronized String getEmgLocal() throws RemoteException {
    return emgLocal;
}

@Override
public synchronized List<String> getControlledEdges(String tlsId)
        throws RemoteException {
    try {
        List<Edge> edges = sim.getControlledEdges(tlsId);
        List<String> aux =  new ArrayList<String>();
        for(Edge e : edges) aux.add(e.getID());
        return aux;
    } catch (IOException e1) {
        e1.printStackTrace();
        return null;
    }
}
```

Figure 10: Java Traffic.java.

```java
@Override
public synchronized Map<String, List<Integer>> getControlledEdgeIds(String tlsId)
        throws RemoteException {
    try {
        List<Edge> edges = sim.getControlledEdges(tlsId);
        Map<String, List<Integer>> edgeMap = new HashMap<String, List<Integer>>();
        for(Edge e : edges) {
            List<Integer> indexes = sim.getLanesIndexes(tlsId, e.getID());
            edgeMap.put(e.getID(), indexes);
        }
        return edgeMap;
    } catch (IOException e1) {
        e1.printStackTrace();
        return null;
    }
}
```

Figure 11: Java Traffic.java.

```java
public synchronized int getNumberOfLanes(String tlsId) throws RemoteException {
    try {
        TrafficLight tls = sim.getTrafficLightRepository().getByID(tlsId);
        if(tls == null) return -1;
        return tls.queryReadControlledLanes().get().size();
    } catch (IOException e) {
        e.printStackTrace();
        return -2;
    }
}
```

Figure 12: Java Traffic.java.

```
public synchronized Map<Integer, List<Integer>> getUncrossedLanes(String tlsId)
        throws RemoteException {
    try {
        TrafficLight tls = sim.getTrafficLightRepository().getByID(tlsId);
        if(tls == null) return null;
        ControlledLinks links = tls.queryReadControlledLinks().get();
        ControlledLink[][] matLinks = links.getLinks();
        Map<Integer, List<Integer>> uncrossMap = new HashMap<Integer, List<Integer>>();
        for(int i = 0; i < matLinks.length; i++) {
            Lane iL = matLinks[i][0].getIncomingLane();
            Point2D pI = iL.queryReadShape().get().getCurrentPoint();

            Lane oL = matLinks[i][0].getOutgoingLane();
            Point2D pO = oL.queryReadShape().get().getCurrentPoint();

            double x = oL.queryReadShape().get().getBounds2D().getX()+oL.queryReadShape().ge
            double y = oL.queryReadShape().get().getBounds2D().getY()+oL.queryReadShape().ge
            Point2D pOWH = new Point2D.Double(x, y);

            Point2D p;

            if(pOWH.equals(pO))
                p = new Point2D.Double(oL.queryReadShape().get().getBounds2D().getX(),
                        oL.queryReadShape().get().getBounds2D().getY());
            else p = pOWH;

            List<Integer> aux = new ArrayList<Integer>();

            for(int j = 0; j < matLinks.length; j++) {
                Lane iC = matLinks[j][0].getIncomingLane();
                Point2D pI1 = iC.queryReadShape().get().getCurrentPoint();

                Lane oC = matLinks[j][0].getOutgoingLane();
                Point2D pO1 = oC.queryReadShape().get().getCurrentPoint();

                double x1 = oC.queryReadShape().get().getBounds2D().getX()+oC.queryReadShape
                double y1 = oC.queryReadShape().get().getBounds2D().getY()+oC.queryReadShape
                Point2D pO1WH = new Point2D.Double(x1, y1);

                Point2D p1;

                if(pO1WH.equals(pO1))
                    p1 = new Point2D.Double(oC.queryReadShape().get().getBounds2D().getX(),
                            oC.queryReadShape().get().getBounds2D().getY());
                else p1 = pO1WH;

                if(Tools.intersects(pI, p, pI1, p1)) aux.add(j);
            }
            uncrossMap.put(i, aux);
        }
        return uncrossMap;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

Figure 13: Java Traffic.java.

```java
@Override
public synchronized String getTLState(String tlsId, String edgeId, String fire)
        throws RemoteException {
    try {
        Edge edge = sim.getEdge(edgeId);
        return sim.getTLState(tlsId, edge, fire);
    } catch (IOException e1) {
        e1.printStackTrace();
        return null;
    }
}

@Override
public synchronized int getNumVehicle(String edge) throws RemoteException {
    try {
        List<Vehicle> vs = sim.getVehiclesAtEdge(sim.getEdge(edge));
        if(vs != null) return vs.size();
        return -1;
    } catch (IOException e) {
        e.printStackTrace();
        return -1;
    }
}
```

Figure 14: Java Traffic.java.

```java
@Override
public synchronized int getNumVehicleEntering(String tlsId, String edgeId) throws RemoteExce
    try {
        TrafficLight tls = sim.getTrafficLightRepository().getByID(tlsId);
        ControlledLink[][] ls = tls.queryReadControlledLinks().get().getLinks();
        for(ControlledLink[] m : ls)
            for(ControlledLink lk : m) {
                Lane outL = lk.getOutgoingLane();
                if(outL.queryReadParentEdge().get().getID().equals(edgeId)) {
                    Lane acl = lk.getAcrossLane();
                    List<Vehicle> vs = sim.getVehiclesAtEdge(acl.queryReadParentEdge().get()
                    int nVeh = -1;
                    if(vs != null) nVeh = vs.size();
                    return nVeh;
                }
            }
        return -1;
    } catch (IOException e) {
        e.printStackTrace();
        return -2;
    }
}
```

Figure 15: Java Traffic.java.

```java
@Override
public synchronized int getNumVehicleExiting(String tlsId, String edgeId) throws RemoteExcep
    try {
        TrafficLight tls = sim.getTrafficLightRepository().getByID(tlsId);
        if(tls == null) return -2;
        ControlledLink[][] ls = tls.queryReadControlledLinks().get().getLinks();
        for(ControlledLink[] m : ls)
            for(ControlledLink lk : m) {
                Lane inL = lk.getIncomingLane();
                if(inL.queryReadParentEdge().get().getID().equals(edgeId)) {
                    Lane acl = lk.getAcrossLane();
                    List<Vehicle> vs = sim.getVehiclesAtEdge(acl.queryReadParentEdge().get()
                    int nVeh = -1;
                    if(vs != null) nVeh = vs.size();
                    return nVeh;
                }
            }
        return -1;
    } catch (IOException | NullPointerException e) {
        e.printStackTrace();
        return -2;
    }
}
```

Figure 16: Java Traffic.java.

```java
@Override
public synchronized double getEdgeLength(String tlsId, String edgeId) throws RemoteExceptior
    try {
        TrafficLight tls = sim.getTrafficLightRepository().getByID(tlsId);
        if(tls == null) return -1;
        Edge e = sim.getEdge(edgeId);
        for(Lane l : tls.queryReadControlledLanes().get())
            if(l.queryReadParentEdge().get().equals(e))
                return l.queryReadLength().get();
        return -1;
    } catch (IOException | NullPointerException e) {
        e.printStackTrace();
        return -2;
    }
}

@Override
public synchronized double getTravelTime(String edgeId) {
    try {
        return sim.getTravelTime(edgeId);
    } catch (Exception e) {
        e.printStackTrace();
        return -1;
    }
}
```

Figure 17: Java Traffic.java.

```java
@Override
public synchronized List<Integer> getLanesIndexes(String tlsId, String edgeId)
        throws RemoteException {
    try {
        return sim.getLanesIndexes(tlsId, edgeId);
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}

@Override
public synchronized void setTLState(String tlsId, String state) throws RemoteException {
    try {
        sim.setTLState(tlsId, state);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

@Override
public synchronized void resetTLS(String tlsId) throws RemoteException {
    try {
        sim.resetTLS(tlsId);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figure 18: Java Traffic.java.

```java
@Override
public synchronized List<String> getVehicles(String edgeId) throws RemoteException {
    try {
        Edge e = sim.getEdge(edgeId);
        List<String> aux = new ArrayList<String>();
        for(Vehicle v : sim.getVehiclesAtEdge(e)) aux.add(v.getID());
        return aux;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}

@Override
public synchronized String sendVehicle(String vType, String vId, List<String> edgeIds,
        int travel) throws RemoteException {
    try {
        Route route = sim.getRoute(vId, edgeIds, travel);
        VehicleType vT = sim.getVehicleTypeRepository().getByID(vType);
        int time = sim.getCurrentSimStep()*1000;
        sim.queryAddVehicle().setVehicleData(vId, vT, route, 0, time, 0, 0);
        sim.queryAddVehicle().run();
        return vType;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

Figure 19: Java Traffic.java.

```java
    @Override
    public synchronized String getVehicleEdge(String vehId) throws RemoteException {
        try {
            Edge edge = sim.getCurrentEdge(vehId);
            if(edge != null) return edge.getID();
            return null;
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }

    @Override
    public synchronized String getCurrentState(String tlsId) throws RemoteException {
        try {
            TrafficLight tls = sim.getTrafficLightRepository().getByID(tlsId);
            if(tls == null) return null;
            return tls.queryReadState().get().toString();
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }

    @Override
    public synchronized boolean isEmg() throws RemoteException {
        return(sim.getCurrentSimStep() >= 20);
    }

    @Override
    public int getnumVictims() throws RemoteException {
        return numVictims;
    }
```

Figure 20: Java Traffic.java.

```
{ include("$jacamoJar/templates/common-cartago.asl") }
{ include("$jacamoJar/templates/common-moise.asl") }

/* Initial beliefs and rules */

routers([]).
amb([]).

rtr(amb,a,"J3",V,N) :- V <= 4 & N = V.
rtr(amb,a,"J3",V,N) :- V > 4 & N = 4.

rtr(plc,a,"J11",V,N) :- V <= 20 & N = 1 + math.round(V/4).
rtr(plc,a,"J11",V,N) :- V > 20 & N = 5.

rtr(frf,a,"J1",V,N) :- V <= 15 & N = 1 + math.round(V/5).
rtr(frf,a,"J1",V,N) :- V > 15 & N = 3.

/* Initial goals */

!run.
```

Figure 21: Jason anl.asl.

```
/* Agent plans */

+!run : not state("finished") <-
    runTraffic;
    .wait(200);
    !run
    .
+!run : startTime(ST) <-
    ET = system.time;
    +emgTime(ET-ST);
    saveData((ET-ST)/1000);
    .
+tls(IDS) <-
    !createTLCs(IDS);
    !verifyEmg;
    .
+!createTLCs(IDS) <-
    for(.member(ID, IDS)){
        .create_agent(ID,"/tlc.asl", [agentArchClass("c4jason.CAgentArch")]);
    };
    .
+!verifyEmg : emg(LCL, VTM) <-
    +startTime(system.time);
    !createRTRs(LCL, VTM);
    +state("started");
    .
+!verifyEmg <-
    isEmg;
    .wait(100);
    !verifyEmg;
    .
```

Figure 22: Jason anl.asl.

```
+!createRTRs(LCL, VTM) <-
    +victims(VTM);
    for(rtr(TYP,GRP,BSE,VTM,NUM)) {
        for(.range(IDX,0,NUM-1)) {
            .concat(TYP,GRP,IDX,NME);
            .create_agent(NME,"/rtr.asl",[agentArchClass("c4jason.CAgentArch")]);
            .send(NME,tell,alert(TYP,BSE,LCL,VTM));
            !addRouter(NME,TYP);
        };
        !victimsToCarry(TYP,NUM);
    };
    .
+!addRouter(NME,amb) : routers(RTL) & amb(AMB) <-
    .term2string(TRM,NME);
    -+routers([TRM|RTL]);
    -+amb([TRM|AMB]);
    .
+!addRouter(NME,_) : routers(RTL) <-
    .term2string(TRM,NME);
    -+routers([TRM|RTL]);
    .
+!victimsToCarry(amb,NUM) : victims(VTM) <-
    -+victims(VTM-NUM);
    .
+!victimsToCarry(TYP,NUM)
    .
```

Figure 23: Jason anl.asl.

```
+finish(NME,TYP) <-
    !manageRTR(NME,TYP);
    .
+!manageRTR(NME,amb) : victims(VTM) & VTM \== 0 <-
    .send(NME,tell,getVictim);
    -+victims(VTM-1);
    .abolish(finish(NME,amb));
    .
+!manageRTR(NME,amb) : amb(AMBL) & routers(RTL) <-
    .delete(NME,AMBL,AMB2L);
    -+amb(AMB2L);
    .delete(NME,RTL,RT2L);
    -+routers(RT2L);
    .
+!manageRTR(NME,_) : routers(RTL) <-
    .delete(NME,RTL,RT2L);
    -+routers(RT2L);
    .
+amb([]) : state("started") & routers(RTL) <-
    .send(RTL,tell,endEmg);
    .
+routers([]) : state("started") <-
    -+state("finished");
    .
```

Figure 24: Jason anl.asl.

```
{ include("$jacamoJar/templates/common-cartago.asl") }
{ include("$jacamoJar/templates/common-moise.asl") }

/* Initial beliefs and rules */

priority(amb,_,2).
priority(plc,_,1).
priority(frf,g,2).
priority(frf,b,1).

/* Initial goals */

!start.

/* Agent plans */

+!start <-
    .my_name(V);
    makeArtifact(V, "mas.RTRTools", [], ID);
    focus(ID);
    .
```

Figure 25: Jason rtr.asl.

```
+alert(TYP,BSE,EMG,VTM) : priority(TYP,g,DRG) <-
    .wait(data("carried"));
    +travel(BSE,EMG,1);
    +state("init");
    !route(BSE,EMG,DRG);
    .
+!route(SRC,TGT,DRG) <-
    findPath(SRC,TGT);
    .abolish(subpath(_,_));
    .my_name(RTR);
    -+segment(0);
    !genSubPath(0,7);       //ALPHA
    !request(RTR,DRG,0);
    !sendVehicle(RTR,SRC,TGT,DRG);
    .
+!genSubPath(0,1) : path(LKS) & .length(LKS,LTH) & LTH >= 1 <-
    for(.member(LK,LKS)) {
        .nth(IDX,LKS,LK);
        +subpath(IDX,[LK]);
    };
    .
+!genSubPath(0,ALPHA) : path(LKS) & .length(LKS,LTH) & LTH > (ALPHA-1) <-
    .findall(LK,(.range(IDX,0,(ALPHA-1),1) & .nth(IDX,LKS,LK)),L);
    +subpath(0,L);
    !genSubPath(ALPHA,ALPHA);
    .
+!genSubPath(INT,ALPHA) : path(LKS) & .length(LKS,LTH) & (INT < LTH-(ALPHA-1)) <-
    .count(subpath(_,_),N);
    .findall(LK,(.range(IDX,INT,INT+(ALPHA-2),1) & .nth(IDX,LKS,LK)),L);
    +subpath(N,L);
    !genSubPath(INT+(ALPHA-1),ALPHA);
    .
+!genSubPath(INT,ALPHA) : path(LKS) & .length(LKS,LTH) & (INT < LTH) <-
    .count(subpath(_,_),N);
    .findall(LK,(.range(IDX,INT,LTH-1,1) & .nth(IDX,LKS,LK)),L);
    +subpath(N,L);
    !genSubPath(INT+(ALPHA-1),ALPHA);
    .
+!genSubPath(INT,ALPHA).
```

Figure 26: Jason rtr.asl.

```
+!request(RTR,DRG,SID) : subpath(SID,LKS) & .nth(0,LKS,LK0) & link(TL,_,LK0,_) <-
    for(.member(LK,LKS)) {
        !sendAlert(RTR,TL,LK,DRG);
    };
    .
+!request(RTR,DRG,SID).
+!sendAlert(RTR,SRC,LK,DRG) : link(ANT,TL,LK,D0) <-
    distance(SRC,ANT,DST);
    .send(TL,tell,priority(RTR,LK,DRG,DST+D0));
    .
+!sendVehicle(RTR,SRC,TGT,DRG) : path(LKS) & alert(TYP,_,_,_) & travel(SRC,TGT,NUM) <-
    sendVehicle(RTR,TYP,LKS,NUM);
    .wait(200);
    getPosition(RTR,DRG);
    .
+position("no",_,DRG) : state("init") <-
    .my_name(RTR);
    .wait(200);
    getPosition(RTR,DRG);
    .
+position("no",_,_) : state("started") <-
    //.print("routing ends!");
    !verifyIfPassed("no");
    -+state("check");
    .my_name(RTR);
    !checkTarget(RTR);
    .
+position(LK,DST,DRG) : state("init") <-
    -+state("started");
    -+curLink(LK);
    .my_name(RTR);
    getPosition(RTR,DRG);
    .
+position(LK,-1,DRG) : state("started") <-
    .my_name(RTR);
    .wait(100);
    getPosition(RTR,DRG);
    .
+position(LK,_,DRG) : state("started") <-
    .my_name(RTR);
    !sendPosition(RTR,LK);
    !verifyToRequest(LK,DRG);
    !verifyIfPassed(LK);
    .wait(200);
    getPosition(RTR,DRG);
    .
```

Figure 27: Jason rtr.asl.

```
{ include("$jacamoJar/templates/common-cartago.asl") }
{ include("$jacamoJar/templates/common-moise.asl") }

/* Initial beliefs and rules */

minDistance(MD) :- .findall(DST,link(LK,DST),DL) & .min(DL,MD).

/* Initial goals */

!start.

/* Agent plans */

+!start <-
    .my_name(TL);
    makeArtifact(TL, "mas.TLCTools", [TL], ID);
    focus(ID);
    .
+priority(RTR,LK,DRG,DST) : state("priority",_) <-
    //.print("2 - priority from ",RTR," on ",LK);
    !genRequest(RTR,LK,DRG,DST);
    .
+priority(RTR,LK,DRG,DST) <-
    //.print("1 - priority from ",RTR," on ",LK);
    +state("priority",LK);
    !genRequest(RTR,LK,DRG,DST);
    !assignPriority;
    .
```

Figure 28: Jason tlc.asl.

```
+!genRequest(RTR,LK,DRG,DST) : minimun(D0) & DST < D0 <-
    -request(RTR,_,_);
    +request(RTR,LK,DRG);
    .
+!genRequest(RTR,LK,DRG,DST) <-
    -request(RTR,_,_);
    +request(RTR,LK,DRG*DST);
    .
+!assignPriority : .findall(CP,request(_,_,CP),CPL) &
    .max(CPL,MAX) & request(RTR,LK,MAX) <-
    !prioritize(RTR,LK);
    .
+!prioritize(RTR,LK) : resetState(RS) & green(LK,G) & yellow(LK,Y) <-
    //.print("Prioritize ",RTR," on ",LK);
    .my_name(TL);
    setTLState(TL,RS);
    .wait(1000);
    setTLState(TL,G);
    //-+reseted("no");
//  .at("now +120 s", {+!forceReset(LK)});
    .
+passed(RTR,LK) <-
    //.print("passed ",RTR," ",LK);
    .abolish(present(RTR,LK));
    .abolish(request(RTR,LK,_));
    .abolish(priority(RTR,LK,_,_));
    !resetTL(RTR,LK);
    .
+request(RTR,LK,_) : not state("priority",LK) <-
    //.print("present ",RTR," ",LK);
    !handleConflict(RTR,LK);
    .
```

Figure 29: Jason tlc.asl.

```
//@handleConflict[atomic]
+!handleConflict(RTR,LK) : request(R0,L0,_) & RTR \== R0 & LK \== L0 <-
    //.print("handle 1 ",RTR," ",LK);
    .wait(passed(R0,L0)[source(R0)]);
    !handleConflict(RTR,LK);
    .
+!handleConflict(RTR,LK) <-
    //.print("handle 2 ",RTR," ",LK);
    -state("priority",_);
    +state("priority",LK);
    !prioritize(RTR,LK);
    .
+!resetTL(RTR,LK) : not request(_,_,_) & yellow(LK,Y)
    & .abolish(state("priority",_)) <-
    //-+reseted("yes");
    //.print("Reseting 1 ",RTR," ",LK);
    .my_name(TL);
    setTLState(TL,Y);
    .wait(1000);
    resetTL(TL);
    .abolish(passed(RTR,LK));
    .
+!resetTL(RTR,LK) <-
    //-+reseted("yes");
    //.print("Reseting 2 ",RTR," ",LK);
    .abolish(passed(RTR,LK));
    .
/*+!forceReset(LK) : reseted("no") & yellow(LK,Y) <-
```

Figure 30: Jason tlc.asl.

```java
/**
package mas;

import features.FileManager;

/**
 * @author Emmanuel Katende Dinanga
 *
 */
public class ANLTools extends Artifact {

    private static final String RMI_PATH = "rmi://127.0.0.1:2020/Traffic";
    private static final Object TRAFFIC_LIGHT = "traffic_light";
    private static final String TRAFFIC_EDGE_OUTPUT = "./src/env/sumo_files/edge-output.xm

    private RemoteInterface rmi;
    private TrafficGraph graph;
    private static int victims = 50;

    public ANLTools() {
        try {
            Traffic.main(victims);
            rmi = (RemoteInterface)Naming.lookup(RMI_PATH);
            graph = new TrafficGraph();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @OPERATION
    public void init() {
        List<TNode> nodes = graph.gettNodes();
        List<jason.asSyntax.Term> aux = new ArrayList<Term>();
        for(TNode n : nodes)
            if(n.getType().equals(TRAFFIC_LIGHT))
                aux.add(ASSyntax.createString(n.getId()));
        defineObsProperty("tls", ASSyntax.createList(aux));
    }

    @OPERATION
    public void runTraffic() {
        try {
            rmi.runTraffic();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    @OPERATION
    public void isEmg() {
        try {
            if(rmi.isEmg())
                defineObsProperty("emg", rmi.getEmgLocal(), rmi.getnumVictims());
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
```

Figure 31: Cartago ANLTools.java.

```
    @OPERATION
    public void saveData(Object endTime) {
        saveTrafficVariableData("emg_total_time", endTime);
    }

    private static void saveTrafficVariableData(String fileTitle, Object data) {
        FileManager fm = new FileManager();
        List<String> content = fm.readFile("data/"+fileTitle+".dat");
        content.add(""+victims+" "+data);
        fm.writeFile(fileTitle+".dat", content);
    }

    public static void saveData(int victims) {
        List<Double> ssL = XMLManager.getMeanData(TRAFFIC_EDGE_OUTPUT, "sampledSeconds");
        List<Double> spL = XMLManager.getMeanData(TRAFFIC_EDGE_OUTPUT, "speed");
        List<Double> tmL = XMLManager.getMeanData(TRAFFIC_EDGE_OUTPUT, "traveltime");
        List<Double> dsL = XMLManager.getMeanData(TRAFFIC_EDGE_OUTPUT, "density");
        List<Double> ocL = XMLManager.getMeanData(TRAFFIC_EDGE_OUTPUT, "occupancy");
        double sumSs = 0, sumSp = 0, sumTm = 0, sumDs = 0, sumOc = 0;
        for(Double ss : ssL) sumSs += ss;
        for(int i = 0; i < spL.size(); i++) sumSp += ssL.get(i)*spL.get(i);
        for(int i = 0; i < tmL.size(); i++) sumTm += ssL.get(i)*tmL.get(i);
        for(int i = 0; i < dsL.size(); i++) sumDs += ssL.get(i)*dsL.get(i);
        for(int i = 0; i < ocL.size(); i++) sumOc += ssL.get(i)*ocL.get(i);
        double speed = Tools.round(sumSp/sumSs, 3);
        double travelTime = Tools.round(sumTm/sumSs, 3);
        double density = Tools.round(sumDs/sumSs, 3);
        double occupancy = Tools.round(sumOc/sumSs, 3);

        saveTrafficVariableData("avg_speed", speed);
        saveTrafficVariableData("avg_travel_time", travelTime);
        saveTrafficVariableData("avg_density", density);
        saveTrafficVariableData("avg_occupancy", occupancy);
    }

    public static void main(String[] args) {

    }
}
```

Figure 32: Cartago ANLTools.java.

```
/**
package mas;

import features.Dijkstra;

/**
 * @author Emmanuel Katende Dinanga
 *
 */
public class RTRTools extends Artifact {

    private static final String RMI_PATH = "rmi://127.0.0.1:2020/Traffic";
    private static final String TRAFFIC_NET_FILE = "./src/env/sumo_files/network.net.xml";
    private static final Object TRAFFIC_LIGHT = "traffic_light";
    private RemoteInterface rmi;

    public RTRTools() {
        try {
            rmi = (RemoteInterface)Naming.lookup(RMI_PATH);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @OPERATION
    public void init() {
        defineObsProperty("data", "carried");
    }

    @OPERATION
    public void findPath(Object source, Object target) {
        try {
            TrafficGraph graph = new TrafficGraph();
            List<TEdge> edges = graph.gettEdges();
            for(TEdge e : edges) {
                double d = XMLManager.getLength(TRAFFIC_NET_FILE, e.getId());
                int nv = rmi.getNumVehicle(e.getId());
                graph.setWeight(e.getId(), d);                  //distance weight.
//              graph.setWeight(e.getId(), nv/d);               //density weight.
            }
            TNode na = graph.getNode(""+source);
            TNode nb = graph.getNode(""+target);
            List<TNode> path = Dijkstra.getShortestPath(graph, na, nb);
            List<jason.asSyntax.Term> aux = new ArrayList<Term>();
            for(int i = 0; i < path.size()-1; i++) {
                TEdge e = graph.getEdge(path.get(i), path.get(i+1));
                aux.add(ASSyntax.createString(e.getId()));
                if(hasObsPropertyByTemplate("link", e.getFrom(), null, null, null))
                    getObsPropertyByTemplate("link", e.getFrom(), null, null, null)
                        .updateValues(e.getFrom(), e.getTo(), e.getId(), e.getWeight());
                else defineObsProperty("link", e.getFrom(), e.getTo(), e.getId(), e.getWei
            }
            if(hasObsProperty("path")) getObsProperty("path").updateValue(ASSyntax.createL
            else defineObsProperty("path", ASSyntax.createList(aux));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Figure 33: Cartago RTRTools.java.

```
    @OPERATION
    public void distance(String a, String b, OpFeedbackParam<Double> d) {
        double d1 = 0;
        if(!a.equals(b)) {
            String to = a;
            do {
                String from = to;
                ObsProperty p = getObsPropertyByTemplate("link", from, null, null, null);
                d1 += p.doubleValue(3);
                to = p.stringValue(1);
            }while(!to.equals(b));
        }
        d.set(d1);
    }

    @OPERATION
    public void sendVehicle(Object veh, Object type, Object[] path, int travel) {
        try {
            List<String> route = new ArrayList<String>();
            for(Object o : path) route.add(""+o);
            rmi.sendVehicle(""+type, ""+veh, route, travel);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @OPERATION
    public void getPosition(Object veh, int degree) throws RemoteException {
        Map<String, Double> map = rmi.getVehicleEdgeAndDistance(""+veh);
        if(map == null) {
            if(hasObsPropertyByTemplate("position", null, null,null))
                getObsPropertyByTemplate("position", null, null,null).
                    updateValues("no", null,degree);
            else defineObsProperty("position", "no", null,degree);
        }
        else {
            String edge = map.keySet().iterator().next();
            double dist = Tools.round(map.get(edge), 3);
            if(hasObsPropertyByTemplate("position", null, null,null))
                getObsPropertyByTemplate("position", null, null,null).
                    updateValues(edge, dist, degree);
            else defineObsProperty("position", edge, dist, degree);
        }
    }

}
```

Figure 34: Cartago RTRTools.java.

```java
/**
package mas;

import features.RemoteInterface;

/**
 * @author Emmanuel Katende Dinanga
 *
 */
public class TLCTools extends Artifact {

    private static final String RMI_PATH = "rmi://127.0.0.1:2020/Traffic";
    private static final String TRAFFIC_NET_FILE = "./src/env/sumo_files/network.net.xml";
    private RemoteInterface rmi;

    public TLCTools() {
        try {
            rmi = (RemoteInterface)Naming.lookup(RMI_PATH);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @OPERATION
    public void init(String tls) {
        try {
            int numLane = rmi.getNumberOfLanes(tls);
            Map<String, List<Integer>> edgesMap = rmi.getControlledEdgeIds(tls);
            Map<Integer, List<Integer>> unCrossLanes = rmi.getUncrossedLanes(tls);
            List<jason.asSyntax.Term> aux = new ArrayList<Term>();
            for(String e : edgesMap.keySet()) {
                aux.add(ASSyntax.createString(e));
                String gState = genGreenState(edgesMap.get(e), numLane, unCrossLanes);
                defineObsProperty("green", e, gState);
                defineObsProperty("yellow", e, genYellowState(gState));
//              defineObsProperty("link", e, XMLManager.getLength(TRAFFIC_NET_FILE, e));
            }
            defineObsProperty("resetState", genResetState(numLane));
            defineObsProperty("links", ASSyntax.createList(aux));
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    private String genGreenState(List<Integer> edgeLanes, int numLane,
            Map<Integer, List<Integer>> unCrossLanes) {
        List<Integer> stateIdx = new ArrayList<Integer>();
        for(int l = 0; l < numLane; l++)
            if(edgeLanes.contains(l))stateIdx.add(l);
            else {
                boolean is = false;
                for(Integer eL : edgeLanes)
                    if(unCrossLanes.get(eL).contains(l)) is = true;
                if(!is) stateIdx.add(l);
            }
        return genState(numLane, stateIdx, "G");
    }
```

Figure 35: Cartago TLCTools.java.

```
private String genState(int size, List<Integer> stateIdx, String fire) {
    String state = "";
    for(int f = 0; f < size; f++)
        if(stateIdx.contains(f)) state += fire;
        else state += "r";
    return state;
}

private String genResetState(int lth) {
    String state = "";
    while(lth > 0) {
        state += "y";
        lth--;
    }
    return state;
}

private String genYellowState(String gState) {
    String state = "";
    for(int i = 0; i < gState.length(); i++) {
        if(gState.charAt(i) == 'r') state += 'r';
        else state += 'y';
    }
    return state;
}

@OPERATION
public void setTLState(Object tls, String state) throws RemoteException {
    rmi.setTLState(""+tls, state);
}

@OPERATION
public void resetTL(Object tl) throws RemoteException {
    rmi.resetTLS(""+tl);
}

@OPERATION
public void getTravelTime(String edge, OpFeedbackParam<Double> time) {
    try {
        double t = rmi.getTravelTime(edge);
        time.set(Tools.round(t, 2)*1000);
    } catch (RemoteException e) {
        e.printStackTrace();
    }
}
```

Figure 36: Cartago TLCTools.java.

```
    @OPERATION
    public void getCurrentState(String tls) {
        try {
            String state = rmi.getCurrentState(tls);
            if(hasObsProperty("curState"))
                getObsProperty("curState").updateValue(state);
            else defineObsProperty("curState", state);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    @OPERATION
    public void getNumVehicles(String edge, OpFeedbackParam<Integer> num) {
        try {
            num.set(rmi.getNumVehicle(edge));
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    @OPERATION
    public void setDemand(String edge, double dmd) {
        dmd = Tools.round(dmd, 2);
        if(hasObsPropertyByTemplate("demand", edge, null))
            getObsPropertyByTemplate("demand", edge, null).updateValues(edge, dmd);
        else defineObsProperty("demand", edge, dmd);
    }

    @OPERATION
    public void setWaitingTime(String edge, double wt) {
        wt = Tools.round(wt, 2);
        if(hasObsPropertyByTemplate("wTime", edge, null))
            getObsPropertyByTemplate("wTime", edge, null).updateValues(edge, wt);
        else defineObsProperty("wTime", edge, wt);
    }

}
```

Figure 37: Cartago TLCTools.java.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet href="http://moise.sourceforge.net/xml/os.xsl" type="text/xsl" ?>

<organisational-specification id="masOrg"
    os-version="0.8" xmlns='http://moise.sourceforge.net/os' xmlns:xsi='http://www.w3.org/2001/X
    xsi:schemaLocation='http://moise.sourceforge.net/os
                        http://moise.sourceforge.net/xml/os.xsd'>

    <structural-specification>
        <role-definitions>
            <role id="anl" />
            <role id="rtr" />
            <role id="tlc" />
        </role-definitions>

        <group-specification id="group">
            <roles>
                <role id="anl" min="1" max="1" />
                <role id="rtr" />
                <role id="tlc" />
            </roles>

            <links>
                <link from="anl" to="rtr" type="communication" scope="intra-group"
                    bi-dir="true" />
                <link from="rtr" to="tlc" type="communication" scope="intra-group"
                    bi-dir="true" />
            </links>

            <formation-constraints>
                <compatibility from="anl" to="rtr" type="compatibility" />
                <compatibility from="rtr" to="tlc" type="compatibility" />
            </formation-constraints>
        </group-specification>
    </structural-specification>
```

Figure 38: Moise masOrg.xml.

```
<functional-specification>
    <scheme id="scheme">
        <goal id="manageEmg">
            <plan operator="parallel">
                <goal id="manageTls">
                    <plan operator="sequence">
                        <goal id="createTlcs" />
                        <goal id="handleJunction">
                            <plan operator="parallel">
                                <goal id="switchTl" />
                                <goal id="handleConflicts">
                                    <plan operator="sequence">
                                        <goal id="prioritize" />
                                        <goal id="resetJunction" />
                                    </plan>
                                </goal>
                            </plan>
                        </goal>
                    </plan>
                </goal>
                <goal id="manageRps">
                    <plan operator="sequence">
                        <goal id="verifyEmg" />
                        <goal id="createRtrs" />
                        <goal id="manageRtrs">
                            <plan operator="sequence">
                                <goal id="findBestPath" />
                                <goal id="sendResponder" />
                                <goal id="updatePosition" />
                            </plan>
                        </goal>
                        <goal id="emdEmg" />
                    </plan>
                </goal>
            </plan>
        </goal>

        <mission id="anlMission" min="1" max="1">
            <goal id="manageEmg" />
            <goal id="verifyEmg" />
            <goal id="createTlcs" />
            <goal id="createRtrs" />
            <goal id="manageTls" />
            <goal id="manageRtrs" />
            <goal id="emdEmg" />
        </mission>

        <mission id="rtrMission">
            <goal id="findBestPath" />
            <goal id="sendResponder" />
            <goal id="updatePosition" />
        </mission>

        <mission id="tlcMission">
            <goal id="handleJunction" />
            <goal id="switchTl" />
            <goal id="handleConflicts" />
            <goal id="prioritize" />
            <goal id="resetJunction" />
        </mission>
    </scheme>
</functional-specification>
```

Figure 39: Moise masOrg.xml.

```
    <normative-specification>
        <norm id="convensionAnL" type="obligation" role="anl" mission="anlMission" />
        <norm id="convensionRtr" type="obligation" role="rtr" mission="rtrMission" />
        <norm id="convensionTlc" type="obligation" role="tlc" mission="tlcMission" />
    </normative-specification>

</organisational-specification>
```

Figure 40: Moise masOrg.xml.

```
package features;

import java.rmi.Remote;

/**
 * @author Emmanuel Katende Dinanga
 */
public interface RemoteInterface extends Remote {

    public double getTravelTime(String edgeId) throws RemoteException;
    public String getTLState(String tlsId, String edgeId, String fire) throws RemoteExcepti
    public void setTLState(String tls, String state) throws RemoteException;
    public List<String> getVehicles(String edgeId) throws RemoteException;
    public String getVehicleEdge(String vehId) throws RemoteException;
    public int getNumVehicleEntering(String tlsId, String edgeId) throws RemoteException;
    public int getNumVehicleExiting(String tlsId, String edgeId) throws RemoteException;
    public double getEdgeLength(String tlsId, String edgeId) throws RemoteException;

    public boolean isEmg() throws RemoteException;
    public String getEmgLocal() throws RemoteException;
    public List<String> getControlledEdges(String tlsId) throws RemoteException;
    public Map<String, List<Integer>> getControlledEdgeIds(String tlsId) throws RemoteExcep
    public void runTraffic() throws RemoteException;
    public int getNumVehicle(String edgeId) throws RemoteException;
    public List<Integer> getLanesIndexes(String tlsId, String edgeId) throws RemoteExceptio
    public int getNumberOfLanes(String tls) throws RemoteException;
    public Map<Integer, List<Integer>> getUncrossedLanes(String tls) throws RemoteException
    public String getCurrentState(String tls) throws RemoteException;
    public String sendVehicle(String vId, String vType, List<String> route, int travel) thr
    public int getnumVictims() throws RemoteException;
    public void resetTLS(String tls) throws RemoteException;
    public Map<String, Double> getVehicleEdgeAndDistance(String string) throws RemoteExcept

}
```

Figure 41: Java RMI RemoteInterface.java.