MASTER IN COMPUTER SCIENCE

Developing agents in the RoboCupRescue simulation project

Leconte, Emmeline; Van Peteghem, Hugues

Award date:
2002

Facultés Universitaires Notre-Dame de la Paix, Namur

Institut d'Informatique

DEVELOPING AGENTS IN THE ROBOCUPRESCUE SIMULATION PROJECT

Emmeline LECONTE et Hugues VAN PETEGHEM

Promoteur : M. Schobbens

Maître de stage : M. Tadokoro

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique

Année Académique 2001 - 2002

Abstract. RoboCupRescue is a research project which the purpose is to provide emergency decision support by integration of disaster information, prediction, planning, and human interface. The main objective of this document is to help new agent developers in this project. To do it, we detail the rescue agents conception by specifying in a clarified way the various steps of development. We give also, as example, our source code with complete explanations, what will allow a better assimilation of the specificities imposed by the project. To be complete, this work introduce the notions of agent, multi-agent system and of multi-agent simulation and confronts them with the project.

Keywords: agent, multi-agent system, multi-agent simulation, RoboCupRescue, developing agents, earthquake.

Résumé. RoboCupRescue est un projet de recherche dont le but est de fournir un support de décision d'urgence, par l'intégration d'informations sur les catastrophes, d'outils de prédiction, de planification, ainsi qu'une interface homme-machine. Ce document a pour principal objectif d'aider de nouveaux développeurs d'agents dans ce projet. Pour ce faire, nous détaillons la conception d'agents secouristes en spécifiant de manière précise les différentes étapes de développement. Nous donnons aussi, à titre d'exemple, notre code source accompagné d'explications complètes, ce qui permettra de mieux assimiler les spécificités imposées par le projet. Afin d'être complet, ce travail introduit les notions d'agent, de système multi-agents et de simulation multi-agents et les confronte au projet.

Mots-clés: agent, système multi-agent, simulation multi-agent, RoboCupRescue, développement d'agents, tremblement de terre.

Acknowledgements

To Professor Pierre-Yves Schobbens,
for the supervision of this work;

To Satoshi Tadokoro, the chairman of the RoboCupRescue Comity,
for his welcome within his work team in Kobe;

To Tomoya Nakanishi, Keiko Watanabe, Professor Takamori, Hiroshi Saito,
Hattori Motofumi, Takuhiro Dohta
and the other colleagues of the Engineering Faculty of Kobe's University,
for their hospitality, their kindness and their patience;

To our families and friends (Japanese, Taiwanese, Danish, Welsh, Spanish,
French and Belgian),
for their support during our five years of studies;

To our classmates,
for the unforgettable years within the Computer Science Institute;

And, finally, both of us,
for this incredible year!

Table of contents

I. INTRODUCTION 12

II. MULTI-AGENT SIMULATION 14

1. INTRODUCTION 14
2. ARTIFICIAL INTELLIGENCE 14
3. DEFINITIONS 16
   1) *Agent* *16*
   2) *Multi-agent system* *18*
   3) *Multi-agent simulation* *20*
4. ENVIRONMENT 21
   1) *Definition* *21*
   2) *Properties* *21*
   3) *Action and modelling action* *21*
   4) *Structure of agent* *24*
   5) *Modelling of environments* *26*
5. RELATIONS BETWEEN AGENTS 27
   1) *Interaction* *27*
   2) *Communication* *30*
6. CONCRETE APPLICATIONS 31
   1) *Foreword* *31*
   2) *Electronic commerce* *31*
   3) *Honey bee simulation* *32*
   4) *RoboCup* *32*

III. ROBOCUPRESCUE: A MULTI-AGENT SIMULATION 38

1. INTRODUCTION 38
2. OVERVIEW 40
3. *GIS* 40
4. *VIEWER* 41
5. *COMPONENT SIMULATORS* 43
6. AGENTS 44
7. *KERNEL* 44

8. PROGRESS OF THE SIMULATION 45

IV. DEVELOPING AGENTS 48

1. INTRODUCTION 48
2. AGENT'S LIFE 48
3. LONG UDP 50
1) Protocol 50
2) Packet format 51
3) Packet header list 52
4. CONNECTION 54
1) Schema 54
2) Packets 55
5. WORLD MODELLING 60
1) Introduction 60
2) World 61
3) Civilian 62
4) FireFighterBrigade 63
5) AmbulanceTeam 64
6) PoliceForce 65
7) Road 65
8) Node 67
9) Building 68
10) Refuge 69
11) FireStation 69
12) AmbulanceCentre 70
13) PoliceOffice 70
14) Concretely 70
6. THE ACTIONS 72
1) Move (route) 73
2) Tell (message) 74
3) Say (message, agent_id) 75
4) Extinguish (building_id) 76
5) Clear (road_id) 77
6) Rescue (agent_id) 77
7) Load (agent_id) 78

8) Unload *79*
9) Code example *80*
7. NEXT KERNEL 80
8. CONCLUSION 81

V. OUR AGENTS 84

1. INTRODUCTION 84
2. MATERIAL AND SIMULATION 84
3. PROGRAMMATION LANGUAGE 86
4. METHODOLOGY AND PRESENTATION OF OUR ARCHITECTURE 87
5. KEEPING MAP DATA 90
6. TALKING WITH THE *KERNEL* 93
7. GENERAL IDEAS 94
8. MOVING ALGORITHM 98
1) Introduction *98*
2) The simulated world map = graph *98*
3) The problem *99*
4) Our solution: Dijkstra's algortihm *100*
5) Implementation *101*
9. COMMUNICATION PROTOCOL 103
10. FIRE FIGHTER BRIGADES ALGORITHMS 105
11. AMBULANCE TEAMS ALGORITHMS 107
12. POLICE FORCES ALGORITHMS 109
13. CENTRE ALGORITHMS 110
14. IMPROVEMENTS 113
15. CONCLUSION 114

VI. CONCLUSION 116

VII. BIBLIOGRAPHICAL REFERENCES 118

VIII. ANNEXES 122

Index of figures and tables

Figure 1: Timeline of major AI events. *14*
Figure 2: The structure of an agent. *24*
Table 1: Classification of interaction situations. *28*
Table 2: Main modes of communication in multi-agent systems *30*
Pictures 1: Kobe's earthquake. *38*
Figure 3: Overview of rescue simulator. *40*
Figure 4: 2D viewer. *41*
Figure 5: 3D viewer. *42*
Figure 6: Component simulators. *43*
Figure 7: Communication at the beginning. *45*
Figure 8: Communication among modules. *46*
Table 3: LongUDP header. *50*
Figure 9: Packet format. *52*
Figure 10: Small packet format. *52*
Table 4: KA packet header. *53*
Table 5: AK packet header. *53*
Figure 11: Connection to the kernel. *54*
Figure 12: Receiving sensory information. *55*
Figure 13: AK_CONNECT packet. *56*
Table 6: Agent's types. *56*
Figure 14: KA_CONNECT_OK packet. *57*
Figure 15: KA_CONNECT_ERROR packet. *58*
Figure 16: AK_ACKNOWLEDGE packet. *58*
Figure 17: KA_HEAR packet. *59*
Figure 18: KA_SENSE packet. *59*
Table 7: World properties. *61*
Table 8: Civilian properties. *63*
Table 9: FireBrigade properties. *64*
Figure 19: Road and node on the RoboCupRescue map. *65*
Table 10: Road properties. *66*

Table 11: Node properties. *67*
Table 12: Building properties. *69*
Table 13: KA_CONNECT_OK filling. *71*
Figure 20: AK_MOVE packet. *73*
Figure 21: AK_TELL packet. *74*
Figure 22: AK_SAY packet. *75*
Figure 23: AK_EXTINGUISH packet. *76*
Figure 24: AK_CLEAR packet. *77*
Figure 25: AK_RESCUE packet. *78*
Figure 26: AK_LOAD packet. *79*
Figure 27: AK_UNLOAD packet. *79*
Table 14: Java Vs. C++. *86*
Figure 28: First version of the RescueAgent package architecture. *88*
Figure 29: ObjectProperties package architecture. *91*
Figure 30: Reaction thread. *96*
Table 15: Graph data. *99*
Figure 31: Tell by communication devices. *103*
Figure 32: Fire fighter algorithm. *107*
Figure 33: Ambulance team algorithm. *108*
Figure 34: Police force algorithm. *110*
Figure 35: Communication example. *111*

I. INTRODUCTION

Thanks to their capacities and their various evolutions, computers always help us in our most varied tasks. And through the years, they become integrated more and more profoundly into the daily life. Our questioning, in this document, aims at a very precise part of the computer science, namely the artificial intelligence or, more exactly, the multi-agent systems. Agents are everywhere. People encounter intelligent agents, information agents, mobile agents, personal assistant agents, and other types mainly in the e-commerce or on the Web. Multi-agent systems are suitable for the domains that involve interactions between different people and collective intelligence.

Agent-based technology has already made a fast inroad from highly specialised workshops to mainstream textbooks. Rapid progress has continued to date, resulting in an ever expanding range of underpinning theories, architectural models, engineering methods, implementation frameworks, and tools. Even so, there remain many issues to be investigated and clarified concerning the relations between theoretical models of agents and multi-agent systems on the one hand, and the deployment of implementations based on these models and architectures in real-world applications on the other, including perspectives on the supporting infrastructure and middleware.

But, would it be possible, using their distributed resolution of problems and their capacity to handle difficult interactions, that these multi-agent systems are capable of advising us and of assisting us in teams management and coordination? Would it be also possible that they manage, one day, to replace us? As regards to teams management in real time, the applications of multi-agent systems are less known. But it is due to the fact that the researches in this domain are only rising up and so, there are not yet many applications. However, we will linger on the research project of the RoboCupRescue which simulates the management of heterogeneous rescue teams after an earthquake.

In this way, we went directly in the heart of the project, where it was born: Kobe. It was in 1995, after the terrible earthquake and the disorganisation of the rescue teams which followed, that Mr. Tadokoro, chairman of the RoboCupRescue project, bent over the possibility of developing multi-agent decision support systems for the first-aid work. A few years later, RoboCupRescue was born. We arrived in Kobe about seven years after the

earthquake and our purpose was, as many other research teams, to develop the most efficient agents.

The interested reader will find our source codes and many other interesting stuffs on the Annexes CD.

This document is organised as follow:

- Chapter II: Multi-agent simulation

This chapter is an introductory chapter in which we define the terms of agent, multi-agent system and of multi-agent simulation. You will also find there some application of the multi-agent systems and finally we formalise the RoboCupRescue simulation project with the definitions exposed.

- Chapter III: RoboCupRescue: a multi-agent simulation

Here, we present the RoboCupRescue project, its purpose, its constituents, its current state and its future versions.

- Chapter IV: Developing agents

This chapter is intended for the beginners who enter in the RoboCupRescue as agents developer. This chapter wants to be a complement to the official manual (V0r4) which you can also find in Annexes. This last one had certain gaps which we tried to fill.

- Chapter V: Our agents

Finally, this chapter presents our work made in Japan: our agents. We detail there all our implementation from the low level layer up to the behavioural algorithms of the various agents. This chapter could also be a good starting point for the beginners trying to come into the project.

II. MULTI-AGENT SIMULATION

1. INTRODUCTION

First, we will try to define clearly the concept of agent, multi-agent system and multi-agent simulation. We take as starting point the book of J. Ferber [FER99]: *Multi-Agent Systems, An introduction to distributed Artificial Intelligence*. We are also inspired by R. Jennings and M. Wooldridge [WOO98]: *Agent technology, Foundations, applications and markets*. These two books allow us to introduce the ideas and theories developed about agents and multi-agents simulations.

Knowing some definitions, we will present practical applications of agents and multi-agent systems and simulations. Finally, we will try to define precisely the RoboCupRescue project using the theories exposed.

2. ARTIFICIAL INTELLIGENCE

With the development of the electronic computer in 1941, the technology finally became available to create machine intelligence. The term artificial intelligence was first coined in 1956, at the Dartmouth conference, and since then Artificial Intelligence has expanded because of the theories and principles developed by its dedicated researchers. Through its short modern history, advancement in the fields of AI have been slower than first estimated but progress continues to be made. From its birth, five decades ago, there have been a variety of AI programs, and they have impacted other technological advancements. The Figure 1 shows the major AI events from 1941 till nowadays.

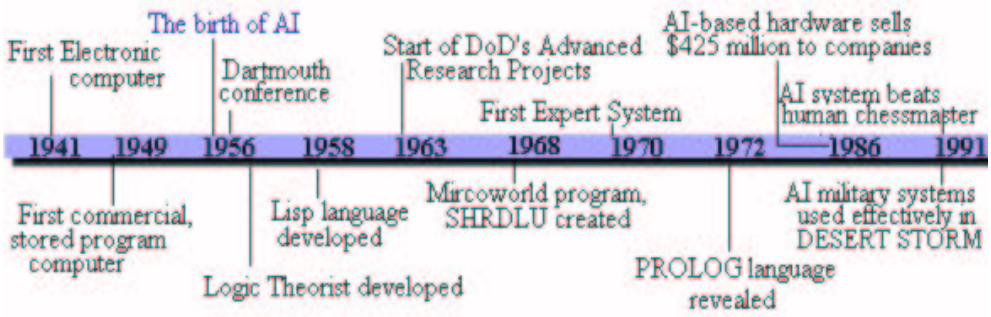


Figure 1: Timeline of major AI events.

In 1998, the first Sony RoboCup-Soccer takes place in Paris at the same time as the world cup of football. The first "functional" humanoid robots are born. And finally in 1999, Sony launches the first model of Aïbo with limited capacities.

Artificial intelligence is a quite young research field. Therefore, we should not be astonished at that the claim of AI to support humans in inaccessible, unpleasant or even hazardous situations by enabling computers to act intelligently, is more a task for next generations that wide-spread reality. Nevertheless, there are a several convincing results to be reported on.

During the eighties, the scientific field which has the most in common with the autonomous agents is planning. This sub-domain of Artificial Intelligence tries to answer to the following question: "What do we do?", in other words, "Which action do we do and in which order?". As an agent is an entity which performs actions, we realise directly the interest of planning. These algorithms gives good results for small and simple problems but, unfortunately, their performances often exponentially decrease when we apply them to bigger problems, as the real world.

Faced with the failure of symbolic AI for some tasks, like walking, another model was set up by R.A. Brooks which is called reactive AI. According to this model, the intelligent behaviour is made of the interaction between other simpler behaviours. Despite its simplicity and its good results, this model receives a lot of reproaches. The main were the difficulty to create an agent with a complex behaviour (too many interactions to be managed) and the difficulty for an agent to learn.

At the early of the nineties, researchers as B. Chaib-Draa and I.A. Ferguson begin to work on the combination of both approaches to obtain an hybrid architecture. In this case, an agent is composed of three layers, at the lowest level of the architecture, we usually find a purely reactive layer, which takes its decisions by basing itself on the environment. The intermediate layer works rather on the knowledge level. And finally, the superior layer takes care of social aspects of the environment, which is taking into account the other agents.

3. DEFINITIONS

1) AGENT

The concept of agent has been the subject of studies in various disciplines during several decades. It was used in the systems with knowledge base, robotics, natural language and of other domains of artificial intelligence. In the literature, there are many definitions of agent. They are alike, but differ according to the type of application for which the agent is conceived.

Ferber considers an agent as a physical or virtual entity:
- which is capable of acting in an environment,
- which can communicate directly with other agents,
- which is driven by a set of tendencies (in the form of individual objectives or of a satisfaction/survival function which it tries to optimise),
- which possesses resources of its own,
- which is capable of perceiving its environment (but to a limited extent),
- which has only a partial representation of this environment (and perhaps none at all),
- which possesses skills and can offer services,
- which may be able to reproduce itself,
- whose behaviour tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representations and communications it receives.

This definition seems to be quite blur to us. It enumerates some properties that seems restrictive. Indeed, just the point about the direct communication disqualifies the RoboCupRescue agents. It is why we think the definition of M. Wooldridge is more convenient: an agent is a computer system, situated in some environment, that is capable of flexible autonomous action in order to meet its design objectives.

Before more discussing about the definitions, it is important to know the two existing schools of thought:
- The first, the 'cognitive' school, considers an agent as follow: a cognitive agent is characterised by a symbolic internal representation of the world which surrounds it. It possesses intentions and tries to reach certain objectives. To do that, it has a representation of the others agents and its environment which can be manipulated and from which it can produce a plan. That means: the sequence of actions which it has to make to reach its determined objective (planning). In this way, each agent has a knowledge base available comprising data required for the carrying out

of its task and for handling interactions with the other agents and with the environment.

- The second, the 'reactive' school, claims that it is not necessary for agents to be individually intelligent for the system to demonstrate intelligent behaviour overall. This schools considers an agent as follows: a reactive agent is defined by the perceptions it has of his environment, by the actions which it can make and by a set of pre-established rules which associate an action to the perceptions.

In front of the difficulty to have a clear and common definition of the concept of agent, Nwana and Ndumu[1] define an agent as referring to a component of software and/or hardware that is capable of acting exactingly in order to accomplish tasks on behalf of its user. They add that it is an umbrella term that covers a range of other more specific agent types. So, it is more enriching to distinct agent characteristics than elaborate a technical and complex definition. (in front of the several dimensions to classify existing software agents) Let us recall the chara of agents given by Nwana and Ndumu:

- *Static* or *mobile*: Agents may be classified by their mobility, their ability to move around some network.

- *Deliberative* or *reactive*: The distinction between the deliberative (cognitive) and reactive agents corresponds to the two existing ways of thought developed above.
  Deliberative agents possess an internal symbolic reasoning model, and they engage in planning and negotiation with other agents in order to achieve their goals.
  Reactive agents, originated from research carried out by Brooks, Agre and Chapman, do not have any internal, symbolic models of their environment, and they act using a stimulus/response type of behaviour by responding to the present state of the environment in which they are embedded (Ferber).

[1] *A brief Introduction to Software Agent Technology*, Nwana, H.S. and Ndumu, D.T. , [WOO98].

- Attributes: *Autonomous* agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.
  Cooperation with other agents is paramount: it is the *raison d'être* for having multiple agents.
  Lastly, agents have to *learn* as they react and/or interact with their external environment; so that with time, their performance increases.

- Roles: We can classify agents by their roles. For example, the information (internet) agents help manage the vast amount of information in the wide area networks like the Internet.

- Hybrid: We also can determine the category of *hybrid* agents that combines two or more agent philosophies in a single agent.

Other attributes of agents can be considered. For example the temporality, the emotional attitudes, … There are others existing classifications, one of the most popular is the Belief Desire Intention (BDI) from Rao and Georgeff [RAO95]. In the BDI, the distinction is inspired by the human reasoning such as we conceive it.

To characterise an agent, will give all its individual characteristics. In section 4) RoboCup page 32, we will describe the agent of the RoboCupRescue simulation project by this way.

But an agent as an individual entity can be limited in many cases. Especially for the distributed applications for which a set of agents putting in common knowledge and competence is more appropriate. This type of application form the multi-agent systems.

2) MULTI-AGENT SYSTEM

Various definitions from different disciplines have been proposed for the term multi-agent system (MAS). According to J. Ferber: the term "multi-agent system" is applied to a system comprising the following elements:

- an environment, E, that is, a space which generally has a volume,
- a set of objects, O. These objects are situated, that is to say, it is possible at a given moment to associate any object with a position in E. These

objects are passive, that is, they can be perceived, created, destroyed and modified by the agents,
- an set of agents, A, which are specific objects ($A \subseteq O$), representing the active entities of the system,
- a set of relations, R, which link objects (and thus agents) to each other,
- an assembly of operations, Op, making it possible for the agents of A to perceive, produce, consume, transform and manipulate object from O,
- operators with the task of representing the application of these operations and the reactions of the world to this attempt at modification, which we shall call the laws of the universe.

In 1998, N.R. Jennings, K. Sycara, and M. Wooldridge have given different meaning to the multi-agent system term, and it is now used for all types of systems composed of multiple autonomous components showing the following characteristics [JEN98]:

- each agent has incomplete capabilities to solve a problem,
- there is no global system control,
- data is decentralised,
- computation is asynchronous.

The definition of Ferber is interesting because the different elements of the MAS are separate in clear and distinct sets. However, the definition of Wooldridge allows passive agents, such as centres in the RoboCupRescue, which can not be formalised with the definition of Ferber.

According to those definitions, a MAS is a distributed system composed of a set of agents. Contrary to AI's systems, which try to simulate the human reasoning, the MAS is conceived as a set of agents interacting according to cooperation, competition, coexistence modes. These systems possess the traditional advantages of the distributed resolution such as modularity, speed or reliability, but they inherit also benefits of the AI which is in each agent.

Although the MAS offers abundant advantages, they also offer many challenges. Indeed, the construction of this type of system contains all the difficulties of distributed systems, but also the flexibility and the complexity of the interactions among agents. As we have seen, the domain of the multi-agents systems is a domain filled with challenges to be surmounted, in other words a domain which is very open for the research.

3) MULTI-AGENT SIMULATION

The precise meaning of the word "simulation" is, even at present, a point of debate, as is the precise definition of "agent" or "multi-agent system". In loose terms, most people understand simulation to be the act of running a program which represents an abstract model in order to study the real system's behaviour. But one thing has to be said: here, the computer has been directly introduced as the suitable partner in the scientific-engineering approach. Though few persons may question this choice, it has to be recognised that other devices like small-scale physical models have also been used and are still in use to assist the researchers in aerodynamic for example. But, in this text, we focus us on the computer-aided modelling and computer simulation. The computer concept of the multi-agents systems is very adapted to perform simulations because it implements independent entities, each of them having its own goal. It is then easy to define the actors of the simulation as computer agents and to implement an abstract model.

So, simulation is a very active branch of computer science which consists of analysing the properties of theoretical models of the surrounding world. Physics, chemistry, biology, ecology, geography and social sciences makes particularly frequent use of simulations to try to explain or forecast natural phenomena. To do this, researchers in these various disciplines construct models of reality and then test their validity by 'running' them on computers.

In theory, the main qualities of multi-agent modelling are its capacity for integration and its flexibility. It is, in fact, possible to integrate within the same model quantitative variables, differential equations and behaviours based on symbolic rules. It is also very easy to integrate modifications, each enhancement of the model being brought about adding behavioural rules which operate at the individual level. In addition, as individuals are different from one another, it is possible to add new types of agents to those already defined. But, as said for the MAS, the multi-agent simulations offer also many challenges beside these advantages.

4. ENVIRONMENT

1) DEFINITION

The environment of a MAS[2] is the common "space" in which the agents of the system are embedded. There is a set of objects easy to handle which have the following particularities:

- situated: at any time, it is possible to determine the position of the object.
- passive or active.

The environment of an agent is composed by the environment of the MAS and the other agents of the system.

2) PROPERTIES

The environment may be described by its properties [BOI01]:

- accessible (or not): the agent can know the complete state of the environment;
- deterministic (or not): the environment is deterministic for a group of agents if its next state is determined by the current state and the actions of the agents;
- episodic (or not): an episodical environment means that the next evolutions do not depend on the actions already carried out;
- static (Vs. dynamic): an environment which does not change while the agent reflects is static;
- discrete (Vs. continuous): if the number of distinct percepts and actions is bounded, the environment is discrete;
- with (or without) any rational adversaries.

3) ACTION AND MODELLING ACTION

Located in its environment, an agent can carry out actions. An action is implemented by a set of effectors, which receive commands from agent. According to this command, to the situation of the agent, to the state of the environment, an action could be carried out. An action can be defined: by the mechanisms implying a modification of the physical environment or like the modification of the physical environment resulting from its application (action) and of the reaction from the environment (co-action). The co-actions are managed by the environment.

[2] There are actually a lot of discussion about the definition of the environment.

There is more than one way of modelling actions and their consequences. Several, more or less mathematical, formalisms are available for taking account of actions, the activities of agents and the evolution of a MAS. Here are some of the formalisms we shall examine:

Action as transformation of global state:

The classic concepts of action used in artificial intelligence are based on an approach involving states and transformations of states.
So we have

- Σ: set of the possible states of the world;
- *op* (args*): action defined as a transition of state (as an operator whose execution produces a new state). args* represents the parameters of the action, for example if (*op* = run), then args can means 50 meters.

Thus, starting from a sate σ_1 of Σ, the execution of the action *op* produces a new state σ_2 of Σ:

σ_2 = Exec (*op* (args*), σ_1), where Exec is the operator of execution for the actions operators, that is, a function of the type: *op* x $\Sigma \rightarrow \Sigma$.

Just like a film strip, each image represents a state of the world, and the actions describe the passage from one image to another (discrete model). So it is the movement of the film that gives the illusion of continuity of movement.

Action as response to influences:

This model is based on the principle of influences and reactions to influences. This model makes use of certain elements from approaches based on the transformation of states. Nevertheless, it takes account of the consequences of the simultaneous actions of agents and model phenomena stemming from interactions between agents.
So we have

- Σ: set of the possible states of the world. (A state is described by a set of atomic formulas);
- Γ: set of possible influences achieved by agents. (An influence is described by a set of atomic formulas);
- Action *op* operator of OP: action whose execution produces a new state (op: $\Sigma \rightarrow \Gamma$). An operator will be represented in the form of a triplet, < *name*, *pre*, *post* >. The application of the operator produces

a set of influences. *pre* describes the conditions which have to be verified for the operator to be applied, and *post* the set of influences which will be directly produced by this operator when it is applied.

The application of an operator can be described by a function, *Exec*, which takes an operator and a current state and gives back an influence:

Exec: OP x $\Sigma \rightarrow \Gamma$

Thus, with the current state σ of Σ, an action *op* of OP and an influence γ of Γ:

Exec (*op*, σ) = γ $\Leftrightarrow$ Exec (< *name*, *pre*, *post* >, σ) = post if *pre*(σ) is verified and {} if not.

The main interest of the influences model lies in the fact that it makes a clear distinction between what properly concern to the agents and the phenomena that take place in the environment. All that is needed is to describe the set of influences which the agent can actually produce and use the environment as an integrator system in which the laws of the universe are the same for all.

Action as computing processes:

Action can also be seen as a set of events produced and consumed by computing. Generally computer science never takes account of the overall state of the world. It conceives the universe as a set of activities which are being carried on in parallel and which are called processes.

We also use the theoretical tools, such as finite-state automata or Petri nets, to formalise the behaviours of the agents and the environment.

Action as local modification:

In local models, action is considered as being a local modification which is propagated along a net of automata. Each action produces only a local disturbance, an alteration which is in contact with, or at any event at a finite distance from, the cause of the action. In this model, the world is made up of a network whose nodes represent 'real' entities and the arcs correspond to the link between these entities (like dependencies, social relationships and so on).

While these models pose numerous problems with regard to the description of agents, they provide an easy way of representing environments, particularly when cellular automata are involved.

Action as physical displacement:

For physics, action is first of all what brings about movement, what changes the dynamic state of a body, that is, an acceleration. Remember the Newton's law: "A body remains at rest or moves with constant velocity in a straight line unless acted upon by force.".

Even if the concept of action as used in physics has had little influence on the world of artificial intelligence, this model is obviously very practical when we want to describe the actions of mobile agents.

Action as command:

In the cybernetic model, action is a command. We are interested in what agent perceives of this environment and in the commands it sends to its actuator organs in order to achieve a goal or satisfy an internal drive. The problem of action then consists in causing variations in a certain number of inputs, known as command variables, in a physical system to obtain specific output variable values. So action is no longer merely a transformation of states of the world or a simple movement, but a complex activity entirely directed towards a goal, which takes account of the reactions of the environment and the corrections to be made to previous actions.

This model is predominantly used to describe the behaviour of the reactive agents.

4) STRUCTURE OF AGENT

After introducing different models of actions we can move on to the description of agents and multi-agent systems. Agents are entities capable of taking account of what surrounds them, which, for purely situated agents, translates into the capacity to *perceive* the environment and to act by *executing* actions which tend to modify the state of the multi-agent system, after having *deliberated* on what should be done. This is schematised in the following figure:

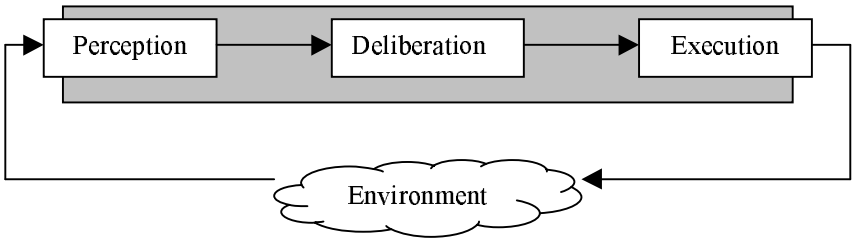


Figure 2: The structure of an agent.

Perception, for an agent, is the quality of being to classify and distinguish states of the world, not only with respect to prominent features of the environment, but also with respect to the actions it is undertaking. The *deliberation*, between the perception and the execution, constitutes the most essential part of an agent. In this part, the objectives, decision making and memory faculties are developed. That is why it is the most complex and the most closely studied.

So, concretely, an agent has different components:

- the physical attributes:
  An agent is characterised by own values (mass, height, colour...) and data relating to the environment (weight, speed...).
- the sensors (*perception*):
  An agent has its own information concerning the environment. This information is acquired by the sensors (intermediary of an interface). Notice that each agent can have different capacities of perception. So, it might be appropriated to allot to each agent its own sensors.
- the effectors (*execution*):
  Without effector an agent has inert behaviour. An effector receives an order of its agent and according to this order, of the situation of the agent and the environment, an action will be perhaps carried out. There is a clear separation between motivation of the agent and the realisation of this motivation.

We can add the following component in the case of the reactive model only:

- the brain:
  The brain is the place where the whole of the processes which, according to the sensors, are charged to order the effector.

5) MODELLING OF ENVIRONMENTS

An agent is environment dependent: once an agent leaves the environment to which it is adapted, it may no longer be considered as an agent. In other words, different agents belong to different environments. In order to describe an agent, it is essential to describe its environment.

So, the environment is the key problem of the modelling of situated agents. There are many possibilities to solve such a problem and we present here a representation developed by J. Ferber [FER99]: BRIC (Basic Representation of Interactive Components). BRIC distinguish clearly two ways to represent the environment:

- The centralised environment
  In this model, the environment is considered as a monolithic block. All the agents have access to the same structure. They produce influences which can be expressed as requests to the environment for actions, and it responds by sending back consumed or perceived elements.

- The distributed environment
  Here the environment is modelled as a set of cells assembled into a network to form a sort of cellular automaton. A distributed environment is made up of a set of cells arranged in a network, like cellular automata. Each cell behaves like a centralised environment in miniature, managing the influences of the agents which are localised on that cell, and sending the consumed marks to the agents that ask for them.

This kind of environment modelling (cells-approach) is largely used in the MAS. And we will see that the distributed model is applicable to the multi-agent simulation project of the RoboCupRescue.

5. RELATIONS BETWEEN AGENTS

1) INTERACTION

Definition

According to Ferber: "An interaction situation as being an assembly of behaviours resulting from the grouping of agents which have to act in order to attain their objectives, with attention being paid to the more or less limited resources which are available to them and to their individual skills.". So, the interactions are related to a series of actions whose consequences exert in return an influence on the future behaviour of the agents. The interactions are not only the consequence of actions carried out by several agents at the same time, but also the element necessary to the constitution of social organisations. It is by the exchanges that they maintain, by engagements which bind them, by the influence which they exert the ones on the others that the agents are social entities and that new functionality can emerge from these systems of mutual actions.

Classification

Each agent can be characterised by three dimensions: its goals, the resources it has and its skills to achieve some tasks. The interactions of the agents of an MAS are justified by interdependence of the agents according to this three dimensions: their goals can be compatible or not ; the agents can wish resources that the others have ; an agent X can have a capacity necessary to an agent Y for the achievement of one of the action plans of Y.

Goals[3]	Resources[4]	Skills[5]	Types of situation	Category
Compatible	Sufficient	Sufficient	Independence	Indifference
Compatible	Sufficient	Insufficient	Simple collaboration	
Compatible	Insufficient	Sufficient	Obstruction	Cooperation
Compatible	Insufficient	Insufficient	Coordinated collaboration	
Incompatible	Sufficient	Sufficient	Pure individual competition	
Incompatible	Sufficient	Insufficient	Pure collective competition	Antagonism
Incompatible	Insufficient	Sufficient	Individual conflicts over resources	
Incompatible	Insufficient	Insufficient	Collective conflicts over resources	

Table 1: Classification of interaction situations.

Following the Table 1, there are eight types of interactions gathered in three main categories: indifference, cooperation and antagonism.

Indifference

In the indifference category, each agent regards the other agents only as components of the environment, as well as all the other components.
The situation of *independence* poses no problem from the multi-agent point of view, and can be summarised as the simple juxtaposition of actions carried out by agents independently, without any effective interaction. For example, people who pass each other in the street, knowing that there is enough room for them to pass.
Simple collaboration consists of the simple addition of skills, requiring no supplementary coordination actions between those involved. This situation is characteristic of the communicating systems in which all the interaction is expressed in the form of the allocation of tasks and the sharing of knowledge.

[3] Represents the compatibility or the incompatibility of the goals of the interactive agents.

[4] Shows if the present resources are sufficient or insufficient for the achieving of the goals of all the interactive agents.

[5] Shows if an agent has the require skills to achieve its own goal.

Cooperation

In the cooperation category, the goal of the agent is not only to maximise its own satisfaction but also to contribute to the success of the group. The agents can exchange information on the environment to increase their individual perceptions, or to transmit their intentions so that the agents can have an idea of what the others do.
Obstruction is characteristic of all situations in which agents get in each other's way in accomplishing their tasks, and do not need one another. An example is the cars on the road.
Coordinated collaboration supposes that the agents have to coordinate their actions to procure the synergic advantages of pooled skills. Coordinated collaboration is the most complex of cooperation situations, since it combines task allocation problems with aspects of coordination shaped by limited resources. Mainly implicated in the industrial activities requiring a distributed approach, such as network control.
In the *pure individual competition* category, the goals are incompatible, agents have to struggle or negotiate in order to achieve them. All the agents have the same resources at their disposal, and they are placed in identical initial situations. Their is no competition about the available resources, so it's pure individual competition. A running race is an example of pure competition.

Antagonism

If the interaction between the agents are antagonist, the goal of each agent is to maximise its own satisfaction, what is made at the expense of the other agents (their goals are incompatible). The agents must thus communicate between them to solve the conflict. This communication takes usually the form of negotiation.
In the *pure collective competition*, when agents do not have sufficient skill, they have to group themselves together into coalitions or associations to be able to achieve their goals. A typical example is the team competitions such as relay races.
In the situation of the *individual conflict over resources*, when resources cannot be shared, we have a typical conflict situation in which an agent wishes to acquire the resources for itself. For example, in the animal world, when animals defend its territory.
The *collective conflicts over resources* combine collective competition with individual conflicts over resources. Coalitions struggle against each other to obtain a monopoly of a good, a territory or a position. The war and other kinds of collective conflicts where the objective is to obtain possession of a territory or a resource are characteristic examples of this.

2) COMMUNICATION

As we have seen in the classification of interaction situations, agents need (sometimes) to communicate to reach their goals. The communication system which binds agents acts as a kind of nervous system which puts in contact sometimes separate individuals. Indeed, the communication increases the perceptive capacities of the agents while enabling them to profit from information and the know-how of the other agents. The communications are essential to the cooperation and it is difficult to conceive a system of cooperating agents if there is not a system to exchange information or to convey requests. In the cognitive systems, the communications are carried out by sending of messages, whereas in the reactive systems, they are the result of the diffusion of a signal in the environment. The communication constitutes one of the fundamental means to ensure the allocation of the functions and the coordination of the actions.

There are distinct communication categories between agents, as detailed in the Table 2:

Types of message	Mode of communication	Routing	Intentionality
point-to-point symbolic message	point-to-point	direct	generally intentional
broadcast symbolic message	general broadcasting	direct	generally intentional
Announcement	point-to-point/broadcasting	noticeboard	generally intentional
Signal	broadcasting	propagation	incidental

Table 2: Main modes of communication in multi-agent systems.

6. CONCRETE APPLICATIONS

1) FOREWORD

Now that we have a better understanding of what the terms 'agent' and 'multi-agent system' mean, the obvious question to ask is: "What do agents have to offer?"

Firstly, multi-agent systems have the ability to solve problems that have hitherto been beyond the scope of automation. The well-known example are open systems (with dynamic structures) or complex systems (which need modularity and abstraction). Secondly, they have the ability to solve problems that can already be solved in a significantly better way. Agent technology provide a better means of conceptualisation or/and implementing a given application.

There are several orthogonal dimensions along which agent applications could be classified such as: industrial applications (process control or air traffic control), commercial applications (electronic commerce detailed just below), medical application (patient monitoring or health care) or games. Here follows three existing multi-agents applications.

2) ELECTRONIC COMMERCE[6]

The SAGE (Smart AGent Environment) project is researching into seamless integration of information distributed over networks. In the SAGE, conversational agents speak in ACL[7] (Agent Communication Language) and cooperate in solving problems focused on retrieval and integration of heterogeneous information.

To put the SAGE into practical uses, it is applied to electronic commerce. Its prototype is called SAGE: Francis, which provides search services for information on commodities. As the result of an experiment with two legacy database applications, Oracle and Access, whose field names and category structures are different, the SAGE: Francis succeeded in integrating

[6] http://context.mit.edu/imediat98/paper4, see also [SUG99].

[7] Communication standard from FIPA (Foundation for Intelligent Physical Agents): The Foundation for Intelligent Physical Agents (FIPA) is an organisation of standardisation which regularly produces specifications aiming a better inter-working between heterogeneous software agents. In 1997, it publishes the FIPA 97 specification whose the second part specifies an ACL (Agent Communication Language). To know more about it: http://www.fipa.org/.

information on commodities and the response time was on average 3 seconds within real-world requirements.

3) HONEY BEE SIMULATION[8]

Honey Bees live in colonies. The colony, or hive, is often considered as a single entity or super-organism but at the same time is made up of tens of thousands of individuals. It is not fully understood how the behaviours of the individuals in a hive contribute to its overall behaviour. This project uses computer simulation to investigate how simple behaviours attributed to individuals could give rise to the patterns of behaviour exhibited at colony level.

In order to simulate the behaviour of bees, the researchers use the Multi-Agent simulation approach which seems to be particularly adapted. Multi-agent simulations give an opportunity to create simulations which are not buried in mathematical definitions. Although all the agents have a simple behaviour, when we begin to consider how they interact in a synchronised time environment, with mathematical definitions, many problems begin to arise. Some of these could greatly alter the outcome of the simulation, even with today's powerful machines, this may not always be possible.

4) ROBOCUP[9]

RoboCup is an international joint project to promote AI, robotics, and related fields. It is an attempt to foster AI and intelligent robotics research by providing standard problem, a football competition and a rescue task, where wide range of technologies can be integrated and examined.

The RoboCup project covers a vast field of applications, in particular as regards robotics and artificial intelligence:

- under development software, the setting of intelligent programs able to develop complex competence in dynamic environment and real time (maintenance of the ball, saving civil under the debris , ...),
- in artificial intelligence, the development of cognitive agents acting in disturbed dynamic environment, able to work in liaison with other agents in coordinated actions, but also able to make autonomous decisions,

[8] http://www.maths.ox.ac.uk/~sumpter/beesim

[9] http://www.RoboCup.org

- in robotics, the development of mobile, autonomous, complex and fast robots. The development of effectors and of reliable receivers in spite of the constraints resulting from the dynamic nature of the environment,
- in image processing, recognition in real time of disturbed information coming from the camera of the agents.

Here follows a description of the two main fields of the RoboCup:

RoboCup Soccer

RoboCup choses to use soccer game as a central topic of research, aiming at innovations to be applied for socially significant problems and industries. The ultimate goal of the RoboCup project is, by 2050, develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer. In order for a robot team to actually perform a soccer game, various technologies must be incorporated including: design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning, robotics, and sensor-fusion. RoboCup is a task for a team of multiple fast-moving robots under a dynamic environment.

RoboCupRescue

RoboCupRescue is a secondary field of the activities of RoboCup. Its main purpose is to provide emergency decision support by integration of disaster information, prediction, planning, and human interface.. RoboCup initiated RoboCupRescue project to specifically promote research in socially significant issues. It is a championship of managing of natural disasters simulation and of use of real robots in such disasters. The championship of simulation concentrates on the strategies of planning and coordination of rescue teams, whereas the competition of robots concentrates on the individual capacities of the robots in the rescue operations, and how these robots can collaborate to achieve specific spots.

Its origin is drawn from the observation of the inefficiency of the current helps in the event of significant natural disaster, like illustrated at the time of the catastrophe of Kobe in Japan. These problems are very close to those with which are confronted the systems multi-agents in strongly dynamic environment. The RoboCupRescue project simulation is an attempt of resolution of these problems. It concentrates mainly on:

- acquisition, accumulation, the relay, selection, analysis, the summary and distribution of information necessary;
- the informative support necessary to the decision;
- distribution of the systems to increase their reliability and their robustness;
- operational continuity, of the normal conditions to the emergencies.

The goal of the RescueCup is similar in spirit with that of RoboCup: in 2050, human technology will have to be able to set up teams of robots which will save lives in real situation and at the time of a so great catastrophe. Currently, the RescueCup is located in margin of RoboCup and utilises teams of simulated agents.

Now we can describe formally the RoboCupRescue simulation project. We plan to analyse the RoboCupRescue simulation project[10] using each point of theory developed.

First of all, we have to give a precise description of the agent. By analysing the typology of agents given by Nwana and Ndumu, we can affirm that:

- The majority of the agents are mobile because they can move around the map. However the centres are static because they are, in fact, building agents.
- The agents are cognitive. They have to possess an symbolic internal representation of the world which surrounds them. They can not share their knowledge about the map. They have specific objectives they try to reach, like extinguish a building in fire.
- An agent is autonomous. It does not need any human intervention and controls his own behaviour.
- The agents needs cooperation to be very effective, for example it is more efficient to extinguish a fire with two or more fire fighter brigades than only one.
- There is no learning system in the RoboCupRescue world.
- We call the agents of the RoboCupRescue **rescue** agents because they all have the common goal to react as rescuers after an earthquake.

At this time the civilians and the rescue agents of the RoboCupRescue simulation project are not very elaborate but, in the future, it is envisaged to develop more human behaviour as the altruism, the selfishness and so on.

[10] More information about the RoboCupRescue simulation project may be very useful to read the following. It is possible to deepen your knowledge about the project by reading the next chapter.

Secondly, the environment of the project can be specify as:

- Not accessible: The agents do not have all the information about the map. They receive, at the beginning of the simulation, the virgin map with the different objects on it, like buildings and roads, but without any information about the real situation (fire, blocked road, …). During the simulation, agents receive the evolution of only a part of the map, which surrounds them.
- Not deterministic: The next state of the environment depends of the actions of the agents **and** the *component simulators*. For example, the state of a burning building depends on the evolution of the simulation (that means whether there are rains, wind, sun, …) determined by the *component simulators*. It depends also of the different initiatives taken by the agents, like extinguish this building.
- Not episodic: The next evolution does not depend only on the realised actions but also of the *component simulators*. In fact, the component simulators manage the natural evolution of the world and the agents after the earthquake. Even if the future situations are influenced by the different actions, like saving a civilian, if this civilian does not have enough air to breathe under the rubble, even if there are tree ambulance teams trying to rescue him, he can be choked.
- Dynamic: The environment is dynamic and changes continuously event when the agent reflects. The updates of the environment are managed by the different simulators who are responsible of the fire evolution, the health evolution of the civilian, … These simulators continue their work even if the agents reflect.
- Discrete: There are a certain number of actions and percepts. The rescue agents and the civilian can not do what they want at this time of the simulation. They have a determine number of simple actions, like move from their position to another place, say a message to someone, load a civilian in an ambulance, …

- Without any adversaries: All the agents in the RoboCupRescue world are interdependent. The management of an earthquake (especially the situation after the earthquake) require the solidarity of all to reduce the damage. It is why the only adversaries in the RoboCupRescue project are the natural phenomena and not other agents. Even if there is a competition to elect the best first-aid worker team, there are no agents adversaries.

Thirdly, there are different ways to model an action in the RoboCupRescue simulation. An action can be seen as *physical displacement*, think about the move around the map. An action is also seen as a *response to influences* because the system takes account of the consequences of the simultaneous actions. For example, when firemen extinguish together a burning building. The actions are *commands* too because to be effective they have to be sent to the kernel.

Fourthly, we can define the interaction between the rescue agents. The category of this interaction is a collaboration and the type of the situation is a coordinated collaboration. In fact, all the rescue agents have the same goal: to bound the damage of the earthquake. However, separately, agent does not have the sufficient resources to act alone. Moreover, it is the sum of the skills of all the rescue agents which allow them to achieve their objective. As example, a fire fighter brigade has to move across the map to extinguish buildings in fire but when roads are blocked, it can not pass through. It is why it needs the help of police forces to clear the roads.

Finally, the type of the messages in the RoboCupRescue, which binds the agents, is the announcement. Each message pass through the kernel. The communication among the agents can be a broadcast or a point to point message.

Now that the RoboCupRescue is clearly defined, the following chapter will detail the project.

III. ROBOCUPRESCUE: A MULTI-AGENT SIMULATION

1. INTRODUCTION

January 17, 1995, 5:47 am: the ground shook in Kobe (in Hanshin-Awaji, Japan). The magnitude of the Hanshin-Awaji Earthquake was 7.2 on the Richter scale. About 2,300,000 people were seriously effected: more than 6.500 citizens died and 43,800 injured. 530,000 buildings were damaged: 104,906 were fully destroyed (mainly wooden houses: 80,000), 144,272 half destroyed and 6,148 were fully burnt. The damage of the basic infrastructures exceeded 100 billion US dollars.

The crush and the suffocation by building destruction were the causes of death in the Hanshin-Awaji Earthquake. The collapse of buildings and the tumble of furniture caused the injury.

Pictures 1: Kobe's earthquake[11].

[11] Source: http://www.eqe.com

Faced with these disastrous consequences of the earthquake some people though about robotics and AI to perform the rescue process. The experiences at the Hanshin-Awaji Earthquake concluded that the following functions were necessary to information systems for disasters:

- Collection, accumulation, relay, selection, summarisation and distribution of necessary information.
- Prompt support for the search and rescue actions.
- Reliability and robustness of the system.

So, some committees have been organised since 1995:

- JSME Robotics and Mechatronics Div. Research Committee which consist of investigation of search and rescue activities in the Hanshin-Awaji Earthquake for the purpose of research and development of rescue robotic infrastructure.
- JSME Research Committee RC-150 which consists of research and development of robotic systems for search and rescue in large-scale disaster.

Others activities were instituted with the aim to promote the rescue research.

On April 30, 1999, a meeting was held at a corner of the hall where people were busy preparing the *RoboCup-Soccer Japan Open '99*. At this meeting, it was decided to make a prototype of the RoboCupRescue simulator by the end of 1999. From that time, many people have joined this project and now, each year, a competition is organised to compare the algorithms of all researchers working on the RoboCupRescue simulation. Last year, in 2001, this competition was held in Seattle (USA) and this year, in June 2002, the competition will be held in Fukuoka (Japan).

The RoboCupRescue project supports the functions for disaster and rescue simulation:

- The period to be simulated is set to the first five hours (300 simulation turns), considering that after that period, survival rate decreases.
- The area of 1.5 km² centred on JR Nagata railway station is selected to be a target for the prototype system. But it's possible to change the map (to put the Los Angeles ward's data for example) and then to have another disaster simulation.
- The number of rescue agents is fixed by the RoboCupRescue committee: five ambulance teams, ten fire-fighter brigades, ten police forces, one ambulance centre, one fire-fighter centre and one police office.

2. OVERVIEW

The RoboCupRescue simulation is built of a number of modules which communicate with each other using a protocol based upon UDP. These modules consist of *kernel*, agents, *component simulators*, *GIS* (Geographical Information System) and *viewer*.

The Figure 3 represents interaction between these various modules but this is a simple overview of the RoboCupRescue simulator. So, in the rest of this chapter, we will take all the different modules one by one and explain them in a more detailed way.

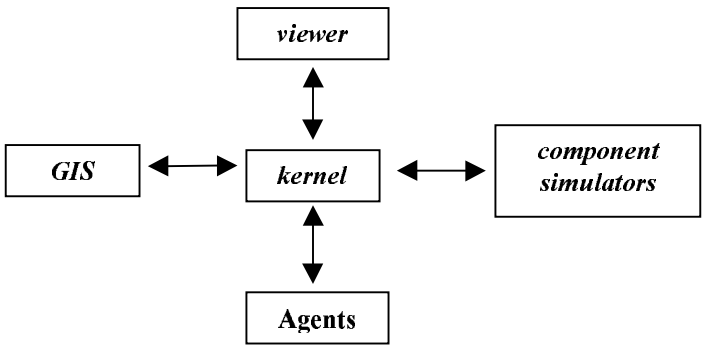


Figure 3: Overview of rescue simulator.

3. *GIS*

The *GIS* (Geographical Information System) module provides the initial configuration of the world, where roads, buildings, nodes and individuals are located at the beginning of the simulation. It manages information necessary to the creation and to the update of the world. The *GIS* contains precise information on each objects world. The world of RoboCupRescue contains many kinds of objects (all these elements will be detailed in the following chapter):

- Buildings: They represent urban constructions in the city. A building can suffer structural damage: from 0 % to 100 %, it can burn, … There are special types of buildings as, for example, the refuge, the fire station, the ambulance centre

and the police office. There are 778 buildings or group of buildings in the current map.

- Roads: They represent a road of the city. It can have different length or width and it can be blocked (from 0 % to 100 %). There are 820 roads (or pieces of road).
- Nodes: They represent the crossing between two (or more) roads or the crossing between a road and a building (an entrance of a building). Totally there are 765 nodes.
- Agents: They may represent all the humans in the city. Agents can be civilians, fire fighters, policemen's or ambulance teams. Totally there are 98 agents on the map.

4. *VIEWER*

The *viewer* visualises the RoboCupRescue simulation. It co-operates with the *GIS* module (for the initialisation) and the *kernel* (for the rest of the simulation) to show the world's state each turn. The Figure 4 shows an example of the 2D *viewer*.

A prototype of a 3D *viewer* is being developed but it is not yet used in the RoboCupRescue simulator (Figure 5).

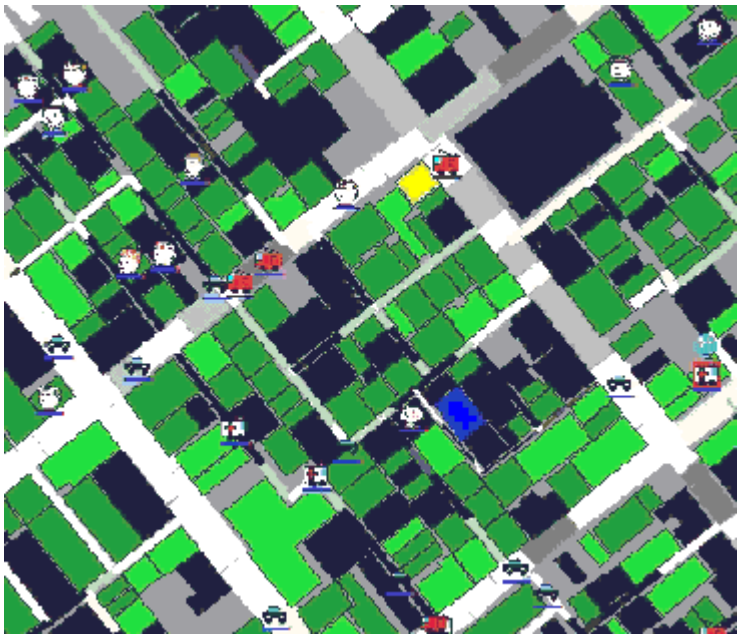


Figure 4: 2D *viewer*.

Agent Developers use mainly the 2D *viewer* which seams to be more useful to watch the simulation result. To understand the map of the Figure 4, it's necessary to know some information:

 : civilian

 : ambulance

 : fire fighter brigade

 : ambulance which load or unload a civilian.

 : police force

The rest of the map is made up of buildings and roads. The buildings may be green, yellow, red or black which gradually represents their burning state. And the roads may be white, grey or black which represents whether they are cleared, partially or totally blocked.

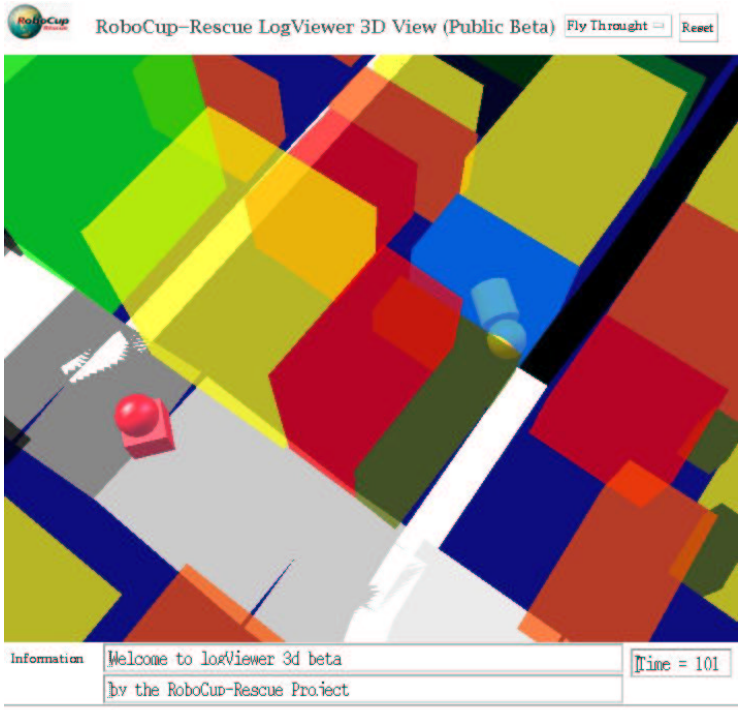


Figure 5: 3D *viewer*.

5. COMPONENT SIMULATORS

Component simulators correspond to various domains, such as earthquakes, fires and traffic jams. The *component simulators*, which are plugged into the system, compute what will happen in the world and what the effects of the individual's actions will be.

For example, the Fire Simulator manages the fire spreading through the building and the intensity of the different fires. Or well the Traffic Simulator cares about the travelling of the different agents through the map.

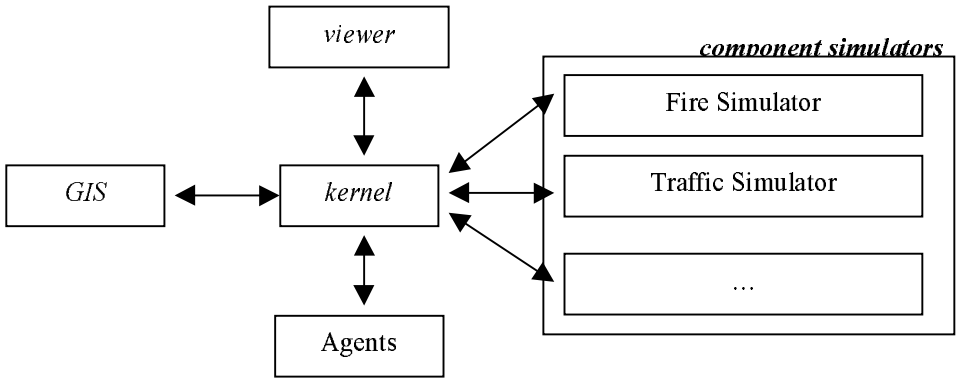


Figure 6: ***Component simulators.***

There is also another simulator, a bit more specific: the miscsimulator (miscellaneous simulator). This simulator takes charge of the agent's status. So, the miscsimulator handles two kinds of things:

I. Commands

- Load: The miscsimulator changes the property *position* of the agent who is being loaded. This property has now the value of the ID of the ambulance team which loaded the agent.
- Unload: The miscsimulator changes the property *position* of the agent who is being unloaded.
- Rescue: The miscsimulator decreases the property *buriedness* of the agent who is being rescued.
- Move: The miscsimulator changes the property *position* of the agent who is moving. The value of this property is the ID of a node or a road or a building.

II. Agent's properties

- Hp: (Health point) The initial value of this property is 10000 and 0 means the agent is dead. The miscsimulator decrease the value of this property when the agent is injured.
- Damage: The miscsimulator puts a certain value on this property when the agent is a victim of fire (40) or a building collapses (400). When the agent is in a refuge, the miscsimulator puts 0 in this property.
- Buriedness: The miscsimulator decreases each turn the value of this property by the number of ambulance teams which are rescuing an agent. When this property is 0, that means the agent is free to move.

6. AGENTS

The agent module controls an intelligent individual that decides its own action according to situations. The individuals are the civilians, the fire fighter brigades, the ambulance teams and the police forces. The fire fighter, police and ambulance centres are agents too but have some particularities: they are static, they can manage the communication between agents, …

Individuals are virtual entities in the simulated world. Their will and actions are controlled by the corresponding agents. The *kernel* supervises the actions of the agents: all the agents decisions pass through the *kernel* (this part is further developed in the next chapter).

Currently, an agent is a unit like a family or a fire brigade in order not to make the simulation system too large. In future, a single person will be represented by an individual agent.

7. KERNEL

The *kernel* is the central module of the simulation. It controls the simulation process and facilitates information among the modules. It's the brain of the simulation which manages all the modules and their communications in real time. It should be able, in the future, to supervise tens of thousand of modules.

One module is represented by one process; however, it is possible to make a single process represent more than one module by using multiple threads. Particularly, the inter-module communication protocol is designed so that more than one agent module can share a socket. It's essential when the number of agents is large (for the time being, there are more than 100 agents).

Agents cannot communicate directly with each other. The communication between agents has to pass through the *kernel* which accepts messages admissible in the communication protocol. Other modules may communicate directly witch each other; however, for the sake of the modularity, plug-in *component simulators* are expected to communicate with others only by way of the *kernel*.

8. PROGRESS OF THE SIMULATION

All the modules described above are connected through the *kernel*. They communicate information to each other at the begin of the simulation and continuously. The communication system among the modules is divided in two parts: the initialisation and the progress of the simulation. The Figure 7 shows the communication at the beginning of the simulation.

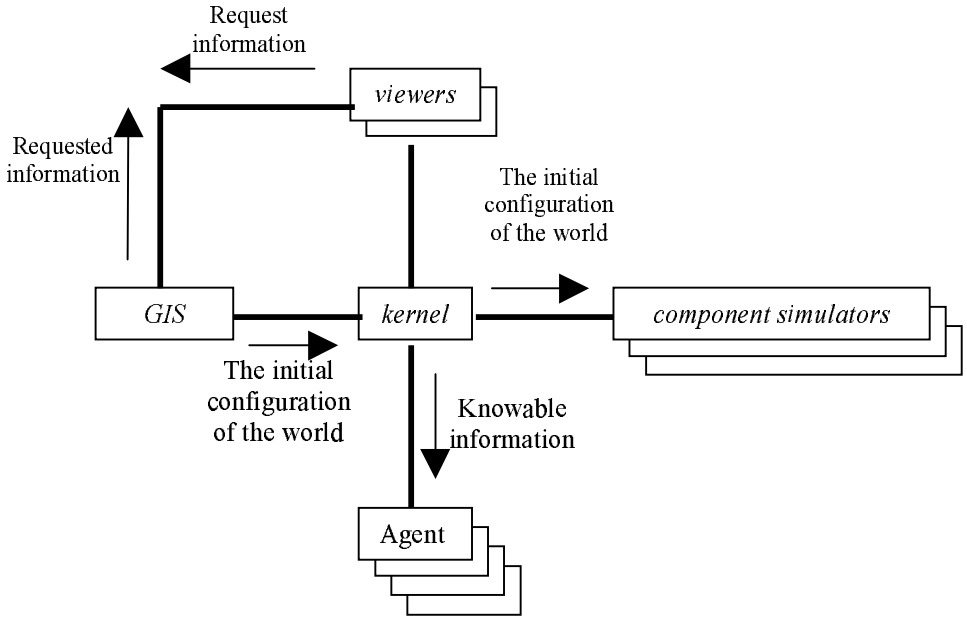


Figure 7: Communication at the beginning.

First of all, the *kernel* receives the initial configuration of the world by the *GIS*. Then with the geographical information it can give the configuration of the world to the *component simulators*. After that it sends the knowable information to the agents; the agent can have sensory information only about a part of the map. The *GIS* and the *viewers* pass on information to one another. So before the start time all the modules have the map information essential for the progress of the simulation. The *kernel* then starts the simulation.

In the prototype system, one cycle taking one second of the computer time simulates one minute of the real world time. During the simulation, information goes around the modules in a way described in Figure 8, which represents the simulation cycle.

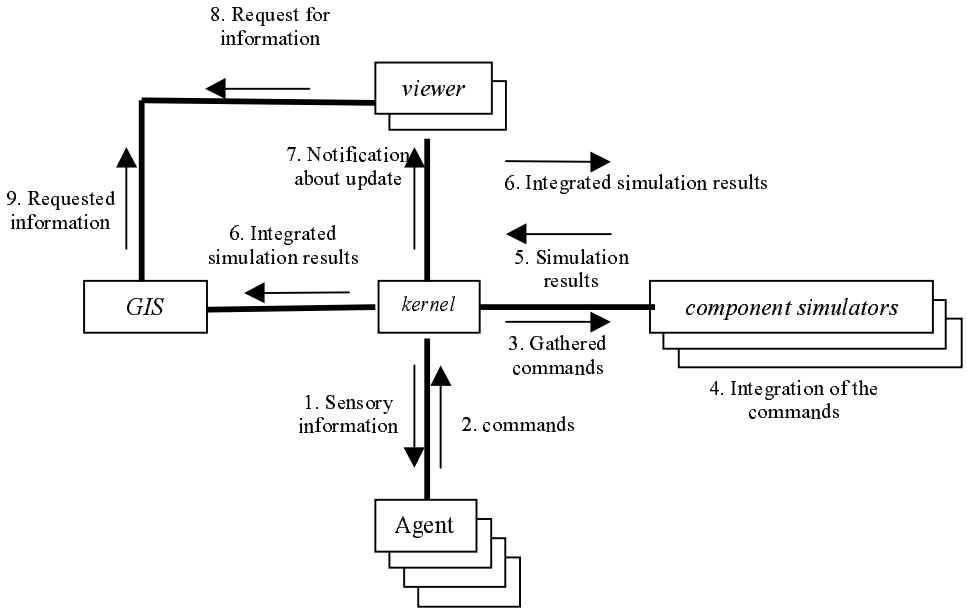


Figure 8: Communication among modules.

The communication among modules follows the steps explained below:

1. At the beginning of each cycle, the *kernel* sends sensory information to each agent module. This sensory information is the visual and hearing information that the agent can sense in the simulated world at that time.

2. With the support of the sensory information the agent module decides what actions the individual should take. When the agent module makes its decision, it send the commands to the *kernel*.

3. The *kernel* gathers all messages from agent modules, filters the commands and broadcasts them to the *component simulators*. The *kernel* supervises the command ; that means, for example: a dead agent cannot send a command or a fire fighter brigade cannot load a civilian!
4. The *component simulators* individually compute how the world will change based upon its internal status and the commands received from the *kernel*. It integrates the actions of the agent module.

5. The *component simulators* have to send back the results of the integration of the agents' actions.

6. The *kernel* integrates the results received from the *component simulators* and broadcast them to the *GIS* and the *component simulators*.

7. The *kernel* increases the simulation clock and notifies the *viewers* about the update.

8. The *viewers* request the *GIS* to send the update information of the world. Then, the *viewers* can visually display the information according to various evaluation criteria.

9. The *GIS* keeps track of the simulation results and sends the *viewers* the requested information.

IV. DEVELOPING AGENTS

1. INTRODUCTION

To develop agents behaviours, the RoboCupRescue manual (V0r4)[12] seems to be light. It's why this chapter develops a more practical manual for agent developers in RoboCupRescue simulation project.

This manual contains information that does not appear in the existing manual. It's not a new version but additional information for agent developers. So to be used efficiently it is recommended to know the RoboCupRescue manual (V0r4) and to assimilate all the concepts of the simulation world.

2. AGENT'S LIFE

There are two major steps to develop an agent:

- at a low level: to maintain the connection with the *kernel* (the availability of the socket and the good formatting of the packets, for example: the header)
- at a higher level: to develop the basic algorithms and the behaviours

The first step is rather technical and is developed in the LongUDP and Connection part of this chapter. The life of an agent begins with the connection to the *kernel*. During a handshake protocol, the agent receives all the simulated world information. The agent has to keep this information in an efficient way because of it's rather large size (explanation in the part world modelling of this chapter). When the connection protocol ends the simulation starts. The agent receives each cycle sensory information from the *kernel* which updates the information received during the connection. Then the agent decides upon some commands to make and sends it to the *kernel*.

[12] The manual version 0, revision 4 (V0r4) is available on http://kiyosu.isc.chubu.ac.jp/RoboCup/Rescue/manual-English-v0r4/manual-v0-r4.pdf.

The second step is more algorithmic. Until now the agents are alive in the simulated world and have all the information they need. However they have to make some commands which represent the actions they want to do. So it's essential to develop some basic and behavioural algorithms. The basic ones are essential: it is, for example, the moving algorithm common to all the agents. Above this elementary algorithms, the agents have to react like rescuers and so to develop their first-aid worker behavioural algorithms. The behaviour of the agent can be sophisticated or simple ; the most important is the result: saving life, limiting the fire, limiting the damage, …

The agents already developed[13] can be classified according to:

- programmation language
- the data structure to keep the world information
- the way to move across the map (moving algorithm)
- the way to save life
- the way to extinguish the fire
- the way to clear the road
- the cooperation between the agents
- the communication protocol between the agents
- …

A third layer can be added above the two others detailed above. This third level layer would be a high level layer containing the behavioural algorithms. An algorithmic hierarchy would be introduced and would allow to develop the behavioural algorithms above the basic and the low level algorithms. This will make it possible to distinguish distinct phases to design agents. In fact, once the two lower level layers are developed in a reliable and powerful way, remains the behavioural layer to implement. That would make it possible to develop the behaviours of the agents independently of the basic and low level algorithms and thus to improve those more easily.

[13] Here is the web page which reference other agents in the RoboCupRescue simulation project: http://www.r.cs.kobe-u.ac.jp/robocup-rescue/robocup2001report.html.

3. LONG UDP

1) PROTOCOL

The communication between *the kernel* and the other components (the agents, the *GIS*, …) is mostly done by UDP. But data from the *GIS* is too big, its length can be larger than 64Kbyte that UDP can handle. So, RoboCupRescue provides a new protocol to transmit those big packets: LongUDP.

This protocol uses IP address and port numbers as well as UDP and they are the same in both protocols. LongUDP divides a big packet (which has a length bigger than 64Kbytes) into small parts, adds a 8 bytes header to each part, puts each part and its header into an IP packet and sends it on the network. The LongUDP header has the following format (Table 3).

Offset (in bytes)	Data	Comment
0	0x0008	Magic number.
2	*ID*	ID number of LongUDP packet.
4	number	This number shows where this UDP packet is in the LongUDP packet. This value varies from 0 to total-1.
6	total	Number of UDP packets of a single LongUDP packet.

Table 3: LongUDP header.

Here is a code example of a method which create a small packet header:

```
DataOutputStream tabHaut;
static Short LudpID = new Short((short) 0);

void create_header_Ludp() throws IOException
        {
        tabHaut.writeShort(0x0008);
        synchronized (LudpID)
                {
                tabHaut.writeShort(LudpID.shortValue());
                LudpID = new Short((short) (LudpID.shortValue() + 1));
                }
        tabHaut.writeShort(0x0000);
        tabHaut.writeShort(0x0001);
        }
```

So, a long packet is reassembled by the receiver by
1. collecting all the LongUDP packets with the same *ID*,
2. after receiving *total* packets, sorting them in *number* ascending order,
3. concatenating them without the header part.

Notice that the length of a divided small part is, except for the last one (of number *total*-1), a multiple of four. This length is bigger than eight to be sure to have data besides the header part.

LongUDP (like UDP) is a minimalist transport protocol which leans on the IP service to supply a transport service without connection and not reliable. That means: no guarantee on the correct routing of packages (a package can be lost), no guarantee on the respect for the sequence (a package can "overtake" an other packet emitted before) and finally, no guarantee on the duplication.

The problems of duplication and 'no respect of the sequence' are not very important due to the sequence number of the LongUDP packets. But the last problem of packet lost is a bit different. In fact there are two issues: the first is when a packet is lost during the connection. During this part of the Simulation, there is a special packets sequence to respect (Figure 11, page 54). So it is easy to notice any abnormality, but this lead to the well known two armies problem [BON00]. The other problem appears when a packet is lost during the rest of the Simulation and this is a bit more complicated to handle. On one hand, if an agent does not receive a packet from the *kernel*, it is not annoying. The next turn, the *kernel* will send an other packet with enough information to fill the lacks. And on the other hand, if the *kernel* does not receive a packet from an agent, the reaction of this agent will depend on its behavioural algorithms.

2) PACKET FORMAT

In the RoboCupRescue Simulation, all the packets sent on the network have the same format: the first 20 bytes are used to create the IP header which contains mainly the destination's IP address and the source's IP address. After this IP header comes the LongUDP header, (eight bytes, Table 3), the following bytes are the body of the packet and the last four bytes are used to encode a special value (Header_NULL = 0x00) to specify the end of the packet (Figure 9).

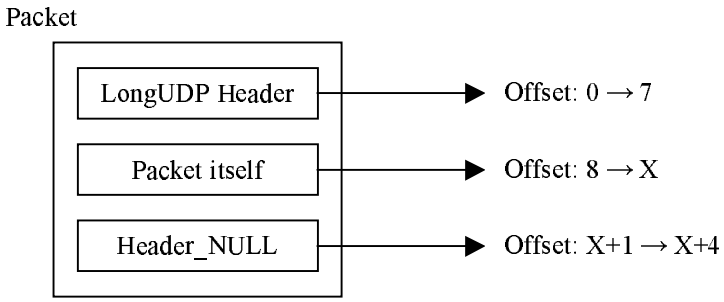


Figure 9: Packet format.

If the length of the packet is lower than 64Kbytes, it is possible to detail the packet format a bit more (this format can also be applied to the first packet of a big packet divided with the LongUDP protocol) (Figure 10).

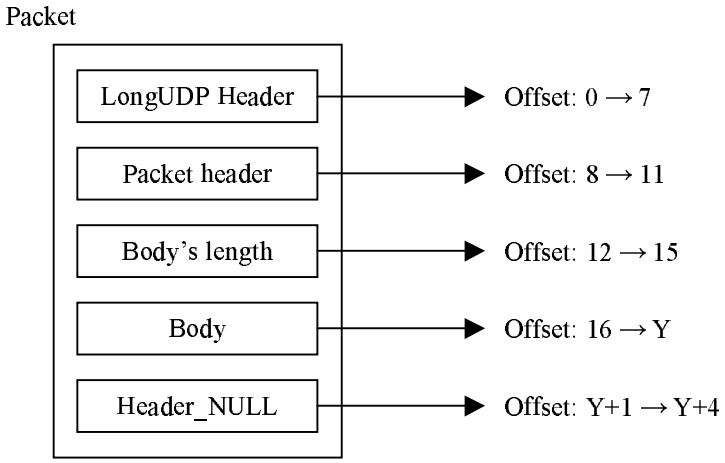


Figure 10: Small packet format.

3) PACKET HEADER LIST

It's possible to divide the packet header in two groups. The first group is formed by the header of all kinds of packets the agent*s* can receive during the simulation. Those packets will come from the *kernel* (K) and are sent to the agents (A) since other communications (eg. direct communication between agents) are forbidden. The name of such header begins with KA (Table 4).

Header's name	Value
KA_CONNECT_OK	0x50
KA_CONNECT_ERROR	0x51
KA_SENSE	0x52
KA_HEAR	0x53

Table 4: KA packet header.

The second group is formed by the header of the packets which come from the agent*s* (A) and are sent to *the kernel* (K). The name of such headers begin with AK (Table 5).

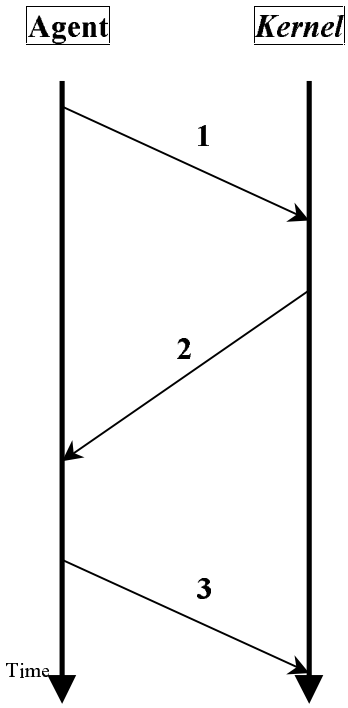
Header's name	Value
AK_MOVE	0x81
AK_LOAD	0x82
AK_UNLOAD	0x83
AK_SAY	0x84
AK_TELL	0x85
AK_EXTINGUISH	0x86
AK_RESCUE	0x88
AK_CLEAR	0x89
AK_CONNECT	0x10
AK_ACKNOWLEDGE	0x11

Table 5: AK packet header.

4. CONNECTION

1) SCHEMA

An agent must connect itself to the *kernel* at the beginning of the simulation. This connection will be done using the IP address and the listening port (usually: 6000) of the *kernel* (Figure 11).



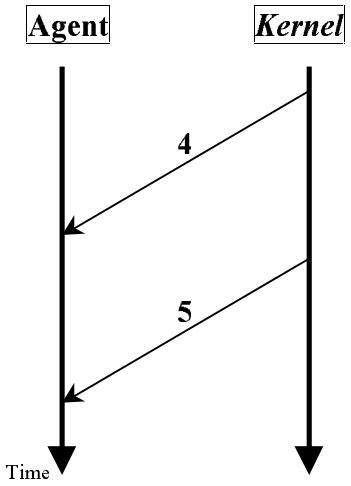
1: AK_CONNECT
First of all, an agent must send an "AK_CONNECT" packet to tell the *kernel* it wants to connect.

2: KA_CONNECT_OK \ ERROR
After receiving an "AK_CONNECT" packet, the *kernel* returns a "KA_CONNECT_OK" when the connection succeeds or returns "KA_CONNECT_ ERROR" in case of failure. The "KA_CONNECT_OK" contains all the simulated world information the agent can have.

3: AK_ACKNOWLEDGE
After receiving a "KA_CONNECT_OK" packet, the agent must send an "AK_ACKNOWLEDGE" packet to be totally connected to the *kernel*.

Figure 11: Connection to the *kernel*.

When all agents are connected (ten fire brigades, ten police forces, five ambulance teams and 3 centres), the simulation may begin. On each cycle, every agent can see (**4**) or hear (**5**) something (Figure 12). It receives from the *kernel* all the sensory information about the surrounding objects. With that information the agent can make commands (such as extinguish a fire for a fire fighter brigade).



4: KA_SENSE
Each agent receives one "KA_SENSE" packet each cycle. In this packet, there is all that this agent can see (the sight is limited at 30 meters).

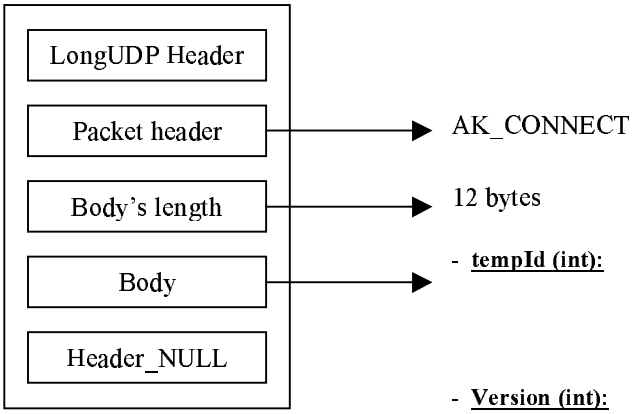
5: KA_HEAR
An agent receives such a packet when somebody (an other agent or a *civilian*) speaks to him (using its 'voice' or telecommunication means).

Figure 12: Receiving sensory information.

2) PACKETS

In this section, we will take, one by one, all the packets sent during the connection between the agent and the *kernel* (KA_CONNECT, KA_ACKNOWLEDGE, KA_CONNECT_OK and KA_CONNECT_ERROR). We will also look at an other kind of packet, those sent carrying sensory information (KA_SENSE and KA_HEAR). We will detail a bit more their content:

- **AK_CONNECT**



- **tempId (int):** Represents the temporary ID given to the newly created agent. This value can be set to any value by corresponding agent module.
- **Version (int):** Represents the version of the *kernel* (for the moment, version is 0).
- **AgentType (int):** Represents the kind of agent that is connecting (Table 6).

Figure 13: AK_CONNECT packet.

AgentType value	Agent's kind
1	Civilian
2	Fire brigade
4	Fire station
8	ambulance team
16	ambulance centre
32	police force
64	police office

Table 6: Agent's types.

Here follows the code example to create a connect packet:

```
VOID CONNECT(INT AGENTTYPE, INT TID)
        {
        (…)
        CREATE_HEADER_LUDP();
        TABHAUT.WRITEINT(AK_CONNECT);
        BYTE[] BODY = NEW BYTE[4 * 3];
        OUTILS.BYTEARRAYCOPY(TID, BODY, 0);
        OUTILS.BYTEARRAYCOPY(0, BODY, 4);
        OUTILS.BYTEARRAYCOPY(AGENTTYPE, BODY, 8);
        TABHAUT.WRITEINT(BODY.LENGTH);
```

```
TABHAUT.WRITE(BODY, 0, BODY.LENGTH);
TABHAUT.WRITEINT(HEADER_NULL);
(…)
}
```

- **KA_CONNECT_OK**

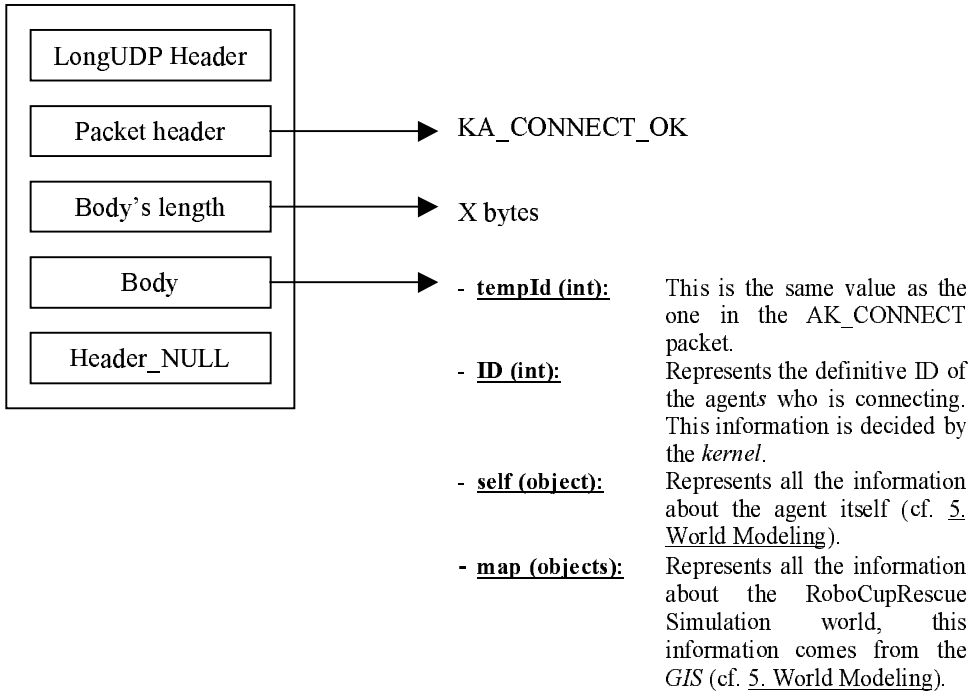


Figure 14: KA_CONNECT_OK packet.

This information is rather large (larger than 64 Kbytes) with regard to all the information about the world that an agent must receive. So, this packet will be divided by the LongUDP protocol and the Figure 13 represents only the first packet of this series of packets.

- **KA_CONNECT_ERROR**

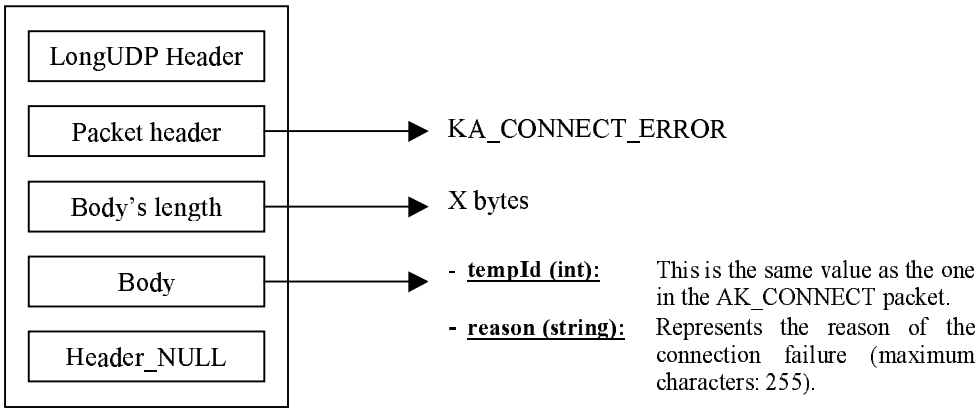


Figure 15: KA_CONNECT_ERROR packet.

- **AK_ACKNOWLEDGE**

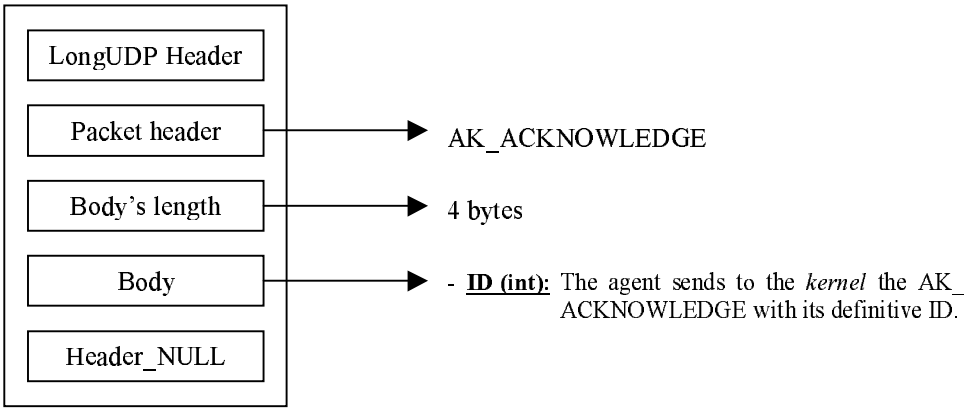


Figure 16: AK_ACKNOWLEDGE packet.

Code example:

```
VOID ACKNOWLEDGE()
        {
        (…)
        CREATE_HEADER_LUDP();
        TABHAUT.WRITEINT(AK_ACKNOWLEDGE);
        BYTE[] BODY = NEW BYTE[4];
        OUTILS.BYTEARRAYCOPY(ID, BODY, 0);
        TABHAUT.WRITEINT(BODY.LENGTH);
        TABHAUT.WRITE(BODY, 0, BODY.LENGTH);
        TABHAUT.WRITEINT(HEADER_NULL);
        }
```

- **KA_HEAR**

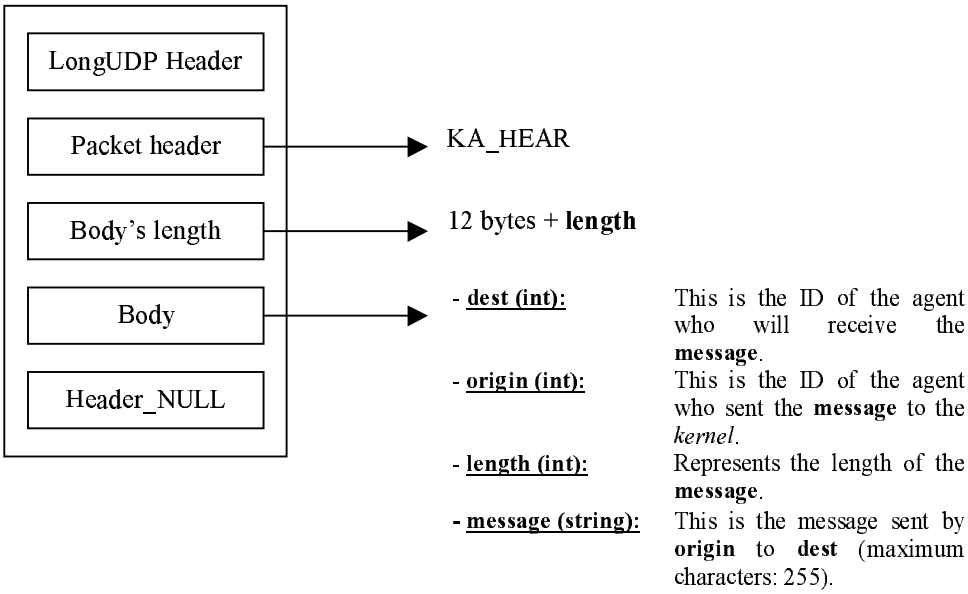


Figure 17: KA_HEAR packet.

- **KA_SENSE**

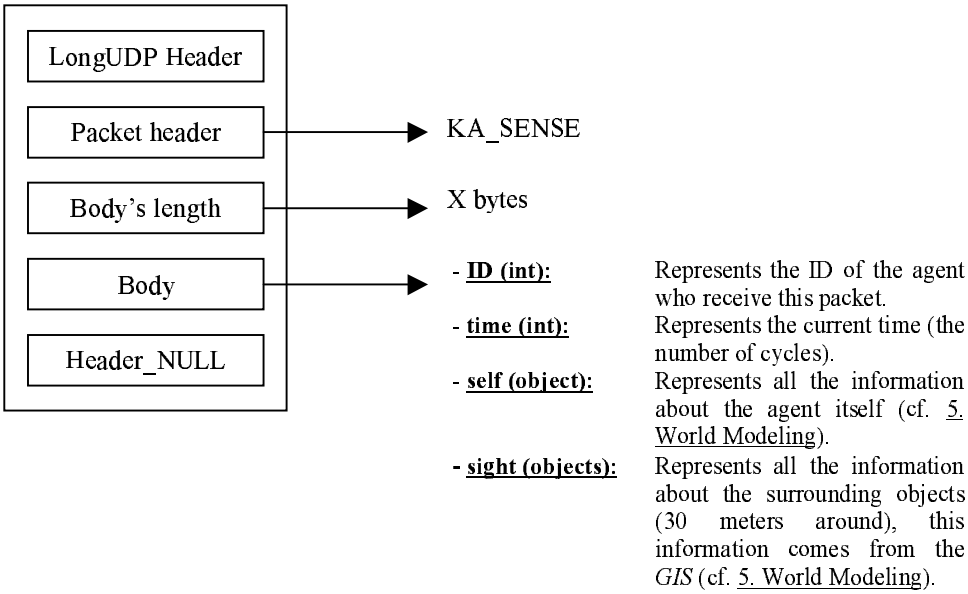


Figure 18: KA_SENSE packet.

5. WORLD MODELLING

1) INTRODUCTION

First, it should be known that the map of the RoboCupRescue world is regarded as a two dimensions plan. Two axes (x and y) make it possible to refer the objects of the world in the map. The measurement unity used is the millimetre. In seek of precision, certain actions and some objects define a property direction which express in second the direction taken by the objet in relation to the y axe.

Let us notice that the *GIS* data format is different of the format of the data which are exchange among the modules of version 0. The *GIS* data is based on the 19 standard coordinate (19_s). 19_s is the coordinate used in Japan Geographical Survey Institute. Normally the unit is the meter, however millimetre is used in RoboCupRescue in order to make the data type in integer. There are a transformation formula[14] which allow to obtain the 19 standard coordinate from the coordinate used in the RoboCupRescue world.

Above this low level representation, there are inter-module rules between all the modules of the RoboCupRescue world to describe all the objects of the map with numbers. So, every object in the simulated world (building, road, civilian and so on) and their characteristics are represented by constants. Moreover, a unique ID is assigned to each existing object of the RoboCupRescue world.

Each module has to be aware to the assigned number because it has to communicate with other modules by using this protocol. Particularly, agents have to keep the map information to act in a efficient way. This information is provided to him by the KA_CONNECT_OK and KA_SENSE messages.

Below, we will detail each kind of object with the following scheme:

- the type number which represent the object.
- an table which contains the description of the properties:
  - Property: the name of the property.
  - Comment: the explanation of the property.
  - Value: description of the real value of the property (minimum and maximum).
  - Property value: the property number which represent the property.

[14] To know more about the *GIS* data format look in the manual (V0r4) on the page 43.

Notice that a certain number object properties do not have any use at this time of the RoboCupRescue project. They describe the simulation world at the best but do not give currently important information for the agents.

2) WORLD

The world object contains some geographical information about the RoboCupRescue environment. Currently, this properties are not very useful to develop agents. However, later it might be useful to use some information like the wind force to improve the quality of the simulation.

- TYPE_WORLD = 208
- descriptions array:

Property	Comment	Value[15]	Property value
startTime	Simulation start time (The time is elapsed minutes from January 1, 1970 at 0:0 AM)	integer: -0x7FFFFFFF ~ 0x7FFFFFFF	29
longitude	The longitude of the new coordinates origin. East longitude is positive and west longitude is negative.	integer: -0x7FFFFFFF ~ 0x7FFFFFFF	30
latitude	The latitude of the new coordinates origin. North latitude is positive and south latitude is negative.	integer: -0x7FFFFFFF ~ 0x7FFFFFFF	31
windForce	Current force of the wind. unit: 0.001 meter/hour.	integer: 0 ~ 0x7FFFFFFF	32
windDirection	Current direction of the wind. The positive direction along y axis is 0 and this value moves to 1295999 (360*60*60-1) seconds in the clockwise rotation direction. The direction is not expressed in term of cardinal points (South, North, East and West).	integer: 0 ~ 1295999	33

Table 7: World properties.

[15] Hexadecimal notation: It's a notation system with base 16. Normally a digit goes from 0 to 9. With the hexadecimal notation, the digit goes from 0 to 15. So, the 16 digits use are: 0 ; 1; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ; A ; B ; C ; D ; E ; F. The hexadecimal system is often use in computer science to code digits with the support of 4 bits.

3) CIVILIAN

The civilians are the basic agents of the RoboCupRescue world. The 72 civilians are given with the simulator. The other agents, the rescue ones, have the same particularities as the civilian, nevertheless, they have additional properties. The civilian are given with the simulator. The 25 rescue agent have to be implement by team developers who want to participate to the competition. The simple civilians have basic behaviour: they shout when they are injured and they walk through the map when they are not under rubble. The rescue agents hear the civilian only if they are distant of less than 30 meter The majority of the civilian are pedestrian but some of them (10) have a car. In the future the behaviour of the civilian will be surely improve to develop some human feature as the altruïsm.

The rescue agents, so the 10 fire fighter brigade, the 5 ambulance team and the 10 police force are detailed below. The civilian have a lot of properties to describe them. This properties are explained here:

- TYPE_CIVILIAN = 232
- descriptions array:

Property	Comment	Value	Property value
position	ID of the object that the civilian is on or in. For example, when a civilian is on a road, position contains the ID of that road (it is the same for building, ambulance, …).	object ID ; 0 (if position = 0 means it is on/in nothing.)	6
positionExtra	When a civilian is on a road, this property signifies the position on the road.: the value is the distance from the head of the road.	integer: 0 ~ x7FFFFFFF. (So the range of the value is from 0 (the civilian is not on a road) to the length of the road.)	7
stamina	Taking actions will decrease the stamina, and the agent cannot take an action that causes the value of the stamina to be negative. It will be restored object each cycle (by *kernel*).	integer: 0 ~ 0x7FFFFFFF	9
hp	Health point, is an amount which determine the health of the civilian. At the beginning, hp is 10000. During the	integer: 0 ~ 0x7FFFFFFF (if hp = 0, that means the civilian	10

	simulation the *kernel* decreases it by damage to represent the injuries of the civilian (see below in the paragraph 6) Rescue at the page 77). When hp=0, the civilian is dead.	is dead)	
damage	This property is the evaluation of the damage that takes the civilian. This also means the necessity of medical treatment. (more details in the next part about the rescue action)	integer: 0 ~ 0x7FFFFFFF	11
buriedness	This property shows how deep the civilian is buried in the collapse of the buildings. The value is how many people are required to save it from the collapse. (see below in the paragraph 6) Rescue at the page 77)	integer: 0 ~ 0x7FFFFFFF	23
direction	This property shows the direction of the civilian. The positive direction along y-axis is 0, and the value moves to 1295999 = 360*60*60-1 seconds counter clockwise.	integer: 0 ~ 1295999	27
positionHistory	A list of Ids, such as buildings or roads, that is passed during the previous one cycle. The order is chronological.	a list of IDs	207

Table 8: Civilian properties.

4) FIREFIGHTERBRIGADE

The fire fighter brigade are the only agent able to extinguish the fire. The 10 firemen have a truck with a hose to sprinkle the buildings on fire. So, normally, they have to pull the hose to an fire hydrant to have water. However, currently the firemen do not have to find a fire hydrant because they have all the water they want in their truck. It is a simplification for the beginning of the simulation project but later there will have some restrictions about it.

A fire fighter brigade, as the other rescue agents, can communicate by radio device with the other fire fighter brigades and can also communicate with

the other rescue agents through the fire fighter centre. The firemen are able to hear 4 messages each cycle. They hear the messages from the fire fighter centre, from the other fire fighter brigades and from the civilian who are shouting. They can tell or say 4 messages that only the others fire fighter brigades and the fire station can hear.

The firemen have the same properties as the simple civilian adding to two others. The two additional properties of the firemen are useless at this time but are described like if they are available.

- TYPE_FIRE_BRIGADE = 233
- descriptions array: the same as the civilian's one plus two other properties.

Property	Comment	Value	Property value
waterQuantity	This property shows how much water is in the tank.	integer: 0 ~ 0x7FFFFFFF	25
stretchedLength	This property shows how long the hose is pulled to the nearest fire hydrant.	integer: 0 ~ 0x7FFFFFFF	26

Table 9: FireBrigade properties.

5) AMBULANCETEAM

The ambulance team are the only agent able to save live. The 5 ambulance teams are able to do different actions. They can excavate rubble to find people. They can load injured in the ambulance and to bring them to the hospital. At this time, the ambulance are only able to transport one man. They can also unload the injured from the ambulance. The ambulance men are able to save other rescue agents.

Currently, at the beginning of the simulation, the ambulances teams (and the other rescue agents) know exactly where are all the civilians on the map. It simplify a lot the research task even if the agents are able to move during the simulation because the majority of them are under the rubble and so does not move anymore. Moreover, the ambulance team know exactly the health of the civilians in a radius of 30 meters. This two information decrease the credibility of the simulation and would be abolish in the future version of the simulation. The ambulance team does not have any other properties than the civilian.

- TYPE_AMBULANCE_TEAM = 234
- descriptions array: the same as the civilian's one.

6) POLICEFORCE

The police force is the only agent able to clear the road. The 10 police men have a bulldozers and other machines to clear the road blocked by rubbles or collapsed. They are responsible to make the street practicable for other agents. The police forces do not have additional characteristics more than those of the civilian.

- TYPE_POLICE_FORCE = 235
- descriptions array: the same as the civilian's one.

7) ROAD

A road in the RoboCupRescue world represents a road of a piece of a road. The Figure 19 is a representation of the road and node on the map. In the following scheme: each ***R*** represents one road of the map, each ***N*** symbolises one node and each ***B*** is one building of the map.

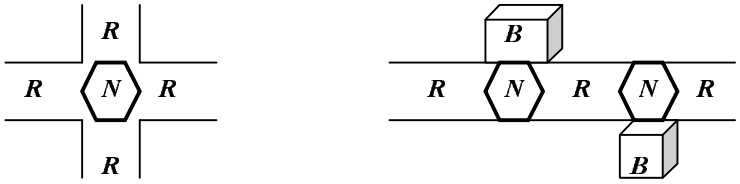


Figure 19: Road and node on the RoboCupRescue map.

The 820 roads of the map have the following properties:

- TYPE_ROAD = 168
- descriptions array:

Property	Comment	Value	Property value
head	ID of an end point of the road.	node ID or building ID	12
tail	ID of the other end point of the road.	node ID or building ID	13
length	The length of the road.	integer (mm): 0 ~ 0x7FFFFFFF	24

roadKind	This property shows the kind of road: - 0x01: elevated road - 0x02: bridge - 0x03: tunnel - 0x04: road for emergency	integer: 0 ~ 0x04	19
carsPassToHead	The number of cars that passed during the previous cycle from tail to head.	integer: 0 ~ 0x7FFFFFFF	34
carsPassToTail	The number of cars that passed during the previous cycle from head to tail.	integer: 0 ~ 0x7FFFFFFF	35
humansPassToHead	The number of humans that passed during the previous cycle from tail to head.	integer: 0 ~ 0x7FFFFFFF	36
humansPassToTail	The number of humans that passed during the previous cycle from head to tail.	integer: 0 ~ 0x7FFFFFFF	37
width	The width of the road.	integer (mm): 0 ~ 0x7FFFFFFF	38
block	The width of the part of the road where cars and humans cannot pass by collapse of buildings, cracks, …	integer (mm): 0 ~ 0x7FFFFFFF	22
repairCost	This property show how many people are required to restore the road.	integer: 0 ~ 0x7FFFFFFF	39
medianStrip	The value is 1 when there is a median strip, otherwise 0.	integer: 0 ~ 0x01	40
linesToHead	The number of traffic lanes from tail to head.	integer: 0 ~ 0x7FFFFFFF	41
linesToTail	The number of traffic lanes from head to tail.	integer: 0 ~ 0x7FFFFFFF	42
widthForWalkers	The width of the part of the road for pedestrians	integer: 0 ~ 0x7FFFFFFF	43

Table 10: Road properties.

8) NODE

A node in the RoboCupRescue models a crossing of roads or a building entrance. The Figure 19 shows the distinct nodes existing in the RoboCupRescue world. So, a node is a crossroad between X distinct roads or an intersection between a road and a building. The X number varies according to the number of road emerging on the node. The 765 nodes in the RoboCupRescue world have the following properties:

- TYPE_NODE = 200
- descriptions array:

Property	Comment	Value	Property value
x	x coordinate	integer (mm): 0 ~ 0x7FFFFFFF	3
y	y coordinate	integer (mm): 0 ~ 0x7FFFFFFF	4
edges	A list[16] of IDs of objects (roads or buildings) connected to this node.	IDs of roads, buildings, …	242
signal	The value is 1 when there are signals, otherwise 0.	Boolean (0 or 1)	44
shortcutToTurn	In the case of left (right) traffic system: a list of short cut for left (right) turn. The numbers are in the order of the edges.	list of integers: 0 ~ 0x7FFFFFFF	128
pocketToTurnAcross	Under the left (right) traffic system: a list of the number of pockets and the length for right (left) turn. The numbers are in the order of the edges.	list of integers: 0 ~ 0x7FFFFFFF	129
signalTiming	A list of the time periods of signal for every road in the edge. The number is a triplet for green, yellow and blue for the right turn. The numbers are in the order of the edges.	list of integers: 0 ~ 0x7FFFFFFF	130

Table 11: Node properties.

[16] A list is a continuation of integer. So, in the case of the edges properties, it is a successive enumeration of the roads or building entrances connected to the node described.

9) BUILDING

The building object in the RoboCupRescue world represents a building or a group of building. In fact, there are the 778 buildings: 768 buildings, 7 refuges, one ambulance centre, one police station and one fire station. A building (included the refuge and the centres) has a lot of particularities to describe him: the number of floors, the entrances and so on. In fact, the refuge and the centres are building agents because they represents the infrastructure of the building and also the personnel who live inside it. The refuge and the centres are detailed below. The type number allow the distinction between all the building kind of the map even if they all ("simple" buildings and agent buildings) have the same properties. These properties are described here after:

- TYPE_BUILDING = 176
- descriptions array:

Property	Comment	Value	Property value
x	x coordinate	integer (mm): 0 ~ 0x7FFFFFFF	3
y	y coordinate	integer (mm): 0 ~ 0x7FFFFFFF	4
floors	The number of floors.	integer: 0 ~ 0x7FFFFFFF	14
buildingAttributes	The value indicates the kind of the building: - 0: wooden house - 1: steel frame house - 2: reinforced concrete house	integer: 0 ~ 0x02	15
ignition	The value is 1 if the Fire Simulator sets fire to, otherwise 0.	Boolean (0 or 1)	48
fieryness	This property shows how much the building is burning: - 0: not burning - 1 → 3: intensity of the fire - 5 or 6: extinguish - 7: totally burned	integer: 0 ~ 0x07	16
brokenness	The value indicates how much the building is collapsed: - 0: no damage	integer: 0 ~ 0x100	17

	- 25: partly damaged - 50: half collapsed - 100: fully collapsed		
entrances	A list of ID's of objects that the entrances of the building connected to.	IDs of the objects (node) connected to the building.	235
buildingShapeID	Figure ID of the building.	integer: 0 ~ 0x7FFFFFFF	49
buildingCode	Structure code.	integer: 0 ~ 0x7FFFFFFF	50
buildingAreaGround	Area of the first ground.	integer: 0 ~ 0x7FFFFFFF	51
buildingAreaTotal	Total floor area.	integer: 0 ~ 0x7FFFFFFF	52
buildingApexes	Coordinates of polygon's vertexes.	list of coordinates	131

Table 12: Building properties.

10) REFUGE

The refuge is a kind of hospital where the agents are in safety. That means that their HP does not decrease anymore. The seven refuges of the RoboCupRescue world can not burn or break down.

- TYPE_REFUGE = 184
- descriptions array: the same as the building's one.

11) FIRESTATION

The fire station is the centre of the fire fighter brigades. The fire station is able to hear the tell and say messages from the agents of the same kind (the firemen so). The fire fighter centre can also tell message to the fire fighter brigades and the other centres. So the centre allows the discussion between the different rescue agents. However, at this time of the competition, with the restriction of the maximum 4 messages to tell and to hear each cycle, the centre become a bottleneck for the communication. This can be applied to the other kind of centre: AmbulanceCentre and PoliceOffice.

- TYPE_FIRE_STATION = 185
- descriptions array: the same as the building's one.

12) AMBULANCECENTRE

- TYPE_AMBULANCE_CENTER = 186
- descriptions array: the same as the building's one.

13) POLICEOFFICE

- TYPE_POLICE_OFFICE = 187
- descriptions array: the same as the building's one.

14) CONCRETELY

So, an object of the simulated world is described with its type, its ID and then its properties. To describe the properties of an object there is a list of couple; the first element of the couples is the type of the property (coded by its number) and the second element is the effective value of the property described for the object.

For example, a building with 7 floors would have the following description: [176 34 14 7 …]. 176 means that the describing object is a building with ID = 34. 14 and 7 are the first couple of the list of properties. 14 means that the number after it represents the number of floors of the building; in this case, the building has 7 floors.

Remember that, at the beginning of the simulation, during the connection sequence, the agent receives the map information in the KA_CONNECT_OK message. At each cycle, the agent receives the same kind of packet: the KA_SENSE, which consists of an updating of the map. But the KA_CONNECT_OK message is much bigger than the KA_SENSE message because it has all the objects of the map in contrast to the KA_SENSE which has only the objects seen by the agent with their new properties.

Here is an example of objects description in the simulated world (the beginning of KA_CONNECT_OK packet):

80 / 344228 / 20 / 2404 / 233 / 2404 / 6 / 892 / 7 / 0 / 27 / 0 / 207 / 0 / 9 / 10000 / 10 / 10000 / 11 / 0 / 23 / 0 / 25 / 26 / 0 / **0** / 208 / 1 / 29 / 0 / 30 / 0 / 31 / 0 / 32 / 0 / 33 / 0 / **0** / 168 / 2 / 12 / 860 / 13 /861 / 24 / 4204 / 19 / 0 / 34 / 0 /35 / 0 / 36 / 0 / 37 / 0 / 38 / 2250 / 22 / 0 / 39 / 0 / 40 / 0 / 41 / 1 / 42 / 1 / 43 / 0 / **0** / 168 / 3 / …

Each object of the world is described in this continuation of figures separated with a **0**.

So the KA_CONNECT_OK packet with the support of the list of the constants can be translated in the following words (the words in italic represents the real value of the property):

80	KA_CONNECT_OK	0	winddirection
344228	length	0	OBJECT
20	tempory ID	168	TYPE_ROAD
2404	ID	2	ID
233	TYPE_FIRE_BRIGADE	12	PROPERTY_HEAD
2404	ID	860	head
6	PROPERTY_POSITION	13	PROPERTY_TAIL
892	position	861	tail
7	PROPERTY_POSITION_EXTRA	24	PROPERTY_LENGTH
0	positionExtra	4204	length
27	PROPERTY_DIRECTION	19	PROPERTY_ROAD_KIND
0	direction	0	kind
207	PROPERTY_HISTORY	34	PROPERTY_CARS_PASS_TO_HEAD
0	history	0	carspasstohead
9	PROPERTY_STAMINA	35	PROPERTY_CARS_PASS_TO_TAIL
10000	stamina	0	carspasstotail
10	PROPERTY_HP	36	PROPERTY_HUMAN_PASS_TO_HEAD
10000	hp	0	humantohead
11	PROPERTY_DAMAGE	37	PROPERTY_HUMAN_PASS_TO TAIL
0	damage	0	humantotail
23	PROPERTY_BURIEDNESS	38	PROPERTY_WIDTH
0	buriedness	2250	width
25	PROPERTY_WATER_QUANTITY	22	PROPERTY_BLOCK
0	water	0	block
26	PROPERTY_STRETCHED_LENGTH	39	PROPERTY_REPAIR_COST
0	stretched	0	repaircost
0	OBJECT	40	PROPERTY_MEDIAN_STRIP
208	TYPE_WORLD	0	medianstrip
1	1 version	41	PROPERTY_LINES_TO_HEAD
29	START_TIME	1	linestohead
0	time	42	PROPERTY_LINES_TO_TAIL
30	LONGITUDE	1	linestotail
0	longitude	43	PROPERTY_WIDTH_FOR_WALKERS
31	LATITUDE	0	widthwalkers
0	latitude	0	OBJECT
32	WIND_FORCE	168	TYPE_ROAD
0	windforce	3	ID
33	WIND_DIRECTION		

Table 13: KA_CONNECT_OK filling.

The KA_CONNECT_OK begins with a description of the agent who receives the packet and the description of the world. After that, all objects of the world are described. In the KA_SENSE, there is first a description of the agent who receives the packet and then, the description of the objects which has been modified (burning, collapsed, …) but only those the agent can have the sensory information about.

So to develop agents it is necessary to retain all information about the map. To be a realistic simulation of the world this memory must be individual (fair play rule).

6. THE ACTIONS

In the RoboCupRescue world, each agent is able to perform different actions, according to its type. Each action must be specified to the *kernel* to be effective. The *kernel* alerts the other simulation components of the actions performed by the agents.

But, for example, how can we specify to the *kernel* that the agent wants to move across the map? To be realised, an action has to be communicated to the *kernel* by using commands. The action command represents the basic communication language between agents and between the agent and the *kernel*.

In the simulated world there are different kinds of agent: the basic ones (the civilian) and the first aid agents (the ambulance team, the fire-fighter brigade and the police force). All agent*s* can do the actions: **move** and **say**. The first aid agents have other specific actions relative to their function. So an ambulance team can **rescue**, **load** and **unload** an other agent ; a police force can **clear** the road ; and a fire fighter can **extinguish** the fire.

In this section, we will detail each of those commands with the same canvas:

Description:	Light description to explain the aim of command.
Actors:	Gives which agent are able to launch this command.
Comments:	Little text to elucidate some subtleties of the command.
Content:	Shows the packet format of the command.

1) MOVE (ROUTE)

- Description: The move action is used to move the agent across the map.
- Actors: Every agent.
- Comments: If the current position of the agent who wants to move is a building, the route will begin with the ID of the building entrance (a node).
  If the actual position is a road the route will begin with the head or the tail of the road (a node).
  Finally if the actual position is a node the route begin with the ID of a road or a building connected to the node.
  Pay attention, it's impossible (using the version 0.31 of the *kernel*) to stop an agent in the middle of a road. So the last ID of a route will always be a building ID or a node ID.
- Content:

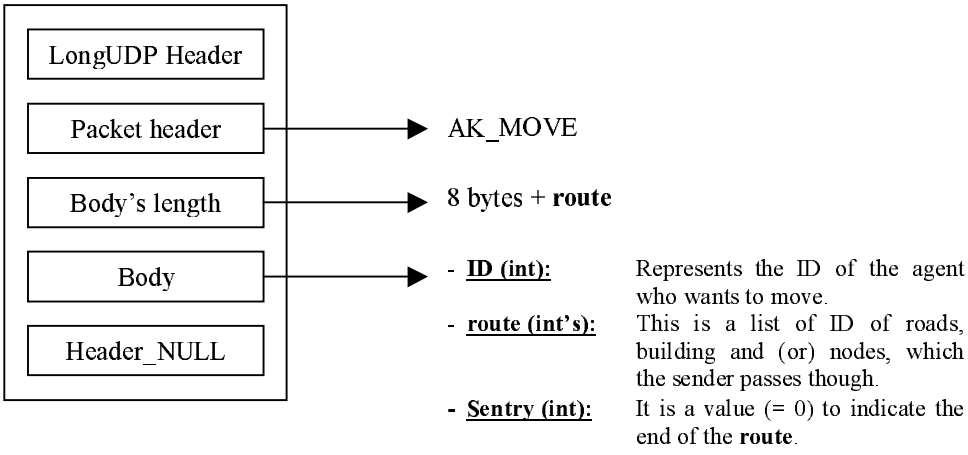


Figure 20: AK_MOVE packet.

2) TELL (MESSAGE)

- Description: The tell command is use to broadcast a message by radio. Every other agent and the centre of the same kind (for example, the ambulance team together and the ambulance centre) will receive this message.

- Actors: All first aid agent (fire brigades, police forces and ambulance teams).

- Comments: Constructing a communication language between each kind of first aid agent is very useful. For example, if one fire brigade needs the help of some colleagues to extinguish a big building, it will use some established messages which are very clear and useful. This special communication between agents can increase the efficiency of the rescue.
  Constructing a communication language between all the first aid agent is useful too. For example, if a fire brigade needs to move to a certain place but the leading there is blocked by collapse buildings, it can ask the police forces for help.
  Notice that the tell from the centres are heard by the other centres too.

- Content:

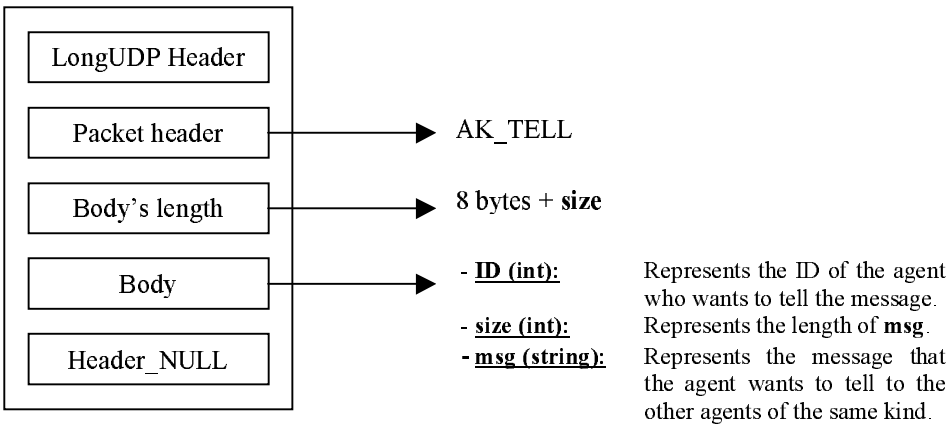


Figure 21: AK_TELL packet.

3) SAY (MESSAGE, AGENT_ID)

- Description: The say command is used to transmit a message to another agent (agent_id) who is within 30 meters with the 'voice'(it's like speaking).

- Actors: Every agent.

- Comments: Say is difficult to use correctly because of two restrictions. The first one is that we must specify the target agent ID. The other is the distance of the availability of the emission (30 meters) which is very little.
  Notice that the civilians use only this command to communicate. Their messages are not normal sentences, they send FIPA ACL messages[17].
  A last comment for the SAY and TELL actions is that any agent can tell or say only 4 sentences in one turn. An other restriction concerns the sentences that an agent is able to hear during each turn: this number is limited to 4.

- Content:

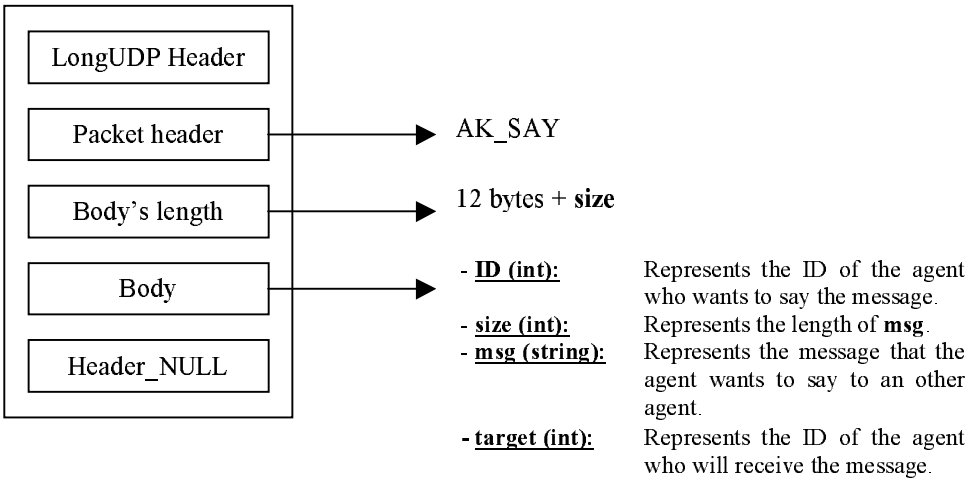


Figure 22: AK_SAY packet.

[17] Communication standard from FIPA (Foundation for Intelligent Physical Agents): http://www.fipa.org.

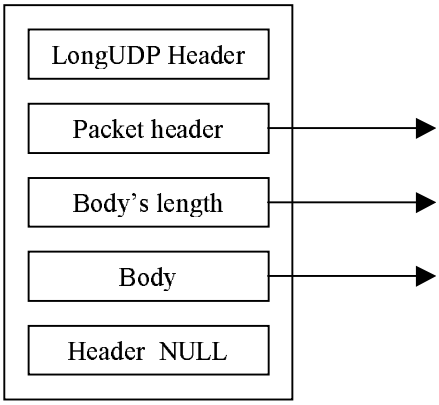
4) EXTINGUISH (BUILDING_ID)

- Description: The extinguish command allows a fire brigade to extinguish a fire which is within a maximum radius of 30 meters.

- Actors: Fire brigades.

- Comments: The extinguish command can only be used if the fire brigade is on a node and if the target building is in the 30 meter radius. The *kernel* 0.31 does not handle the water quantity. So the extinguish command works with a water quantity of 1000.

- Content:



AK_EXTINGUISH

28 bytes

- **ID (int):**	Represents the identifier of the fire brigade who wants to extinguish a fire.
- **targetID (int):**	Represents the ID of the burning building that the fire brigade wants to extinguish.
- **direction (int):**	Represents the direction of the nozzle. This value vary between 0 (along the Y axis) and 1295999 (360*60*60-1).
- **x (int):**	Position of the nozzle (x coordinate)
- **y (int):**	Position of the nozzle (y coordinate)
- **quantity (int):**	This value is equal to 1000.
- **sentry (int):**	It is a value (= 0) to indicate the end of the AK_EXTINGUISH packet.

Figure 23: AK_EXTINGUISH packet.

5) CLEAR (ROAD_ID)

- Description: The clear command allows a police force to restore a blocked road so that cars can pass trough it.
- Actors: police forces.
- Comments: No particular remarks about this command.
- Content:

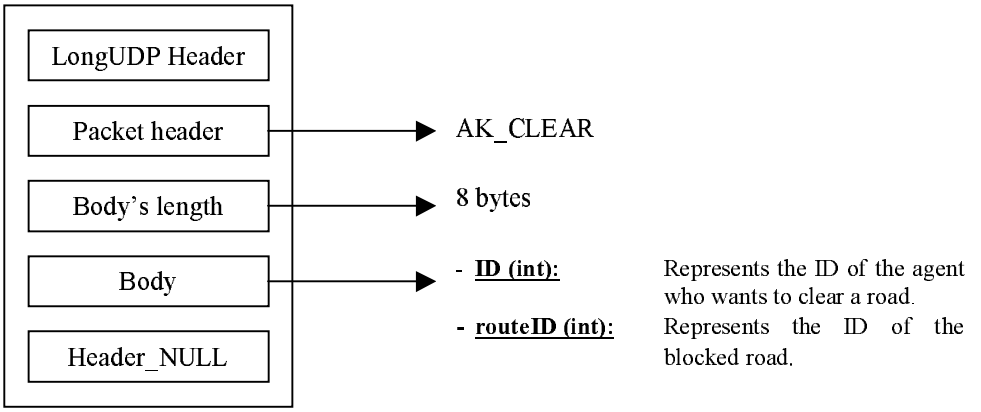


Figure 24: AK_CLEAR packet.

6) RESCUE (AGENT_ID)

- Description: This command is used to rescue buried people. That means to free the civilians from the collapsed buildings.
- Actors: ambulance teams.
- Comments: When an ambulance team rescues a civilian, the property *buriedness* of this civilian is decreased by one. In other words, the property *buriedness* of a civilian shows how many people are required to extricate him from the debris. Notice that rescuing a civilian with two ambulance teams decreased by twice the number of turns necessary to extricate the civilian from the collapse.
  In the same time, the *hp* of the buried civilian decreases each turn of the *damage* value. So, to make the rescues more efficient, it's useful to analyse these fields.

- Content:

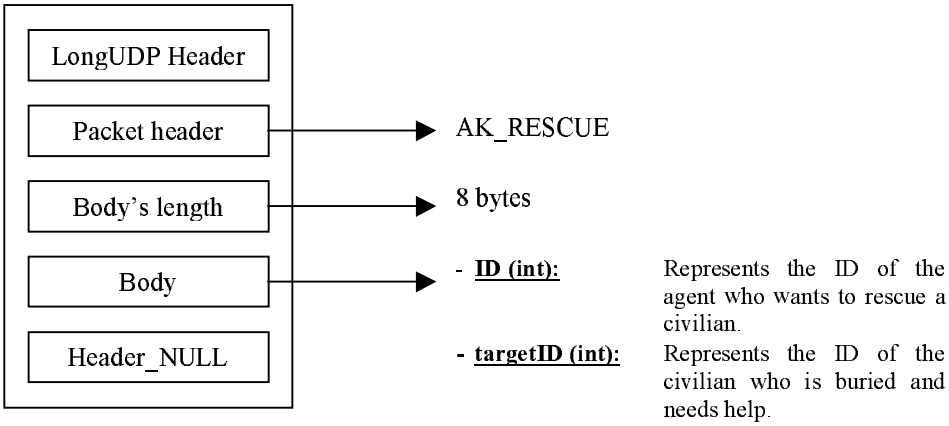


Figure 25: AK_RESCUE packet.

7) LOAD (AGENT_ID)

- Description: This command is used to load an injured civilian into the ambulance.

- Actors: ambulance teams.

- Comments: To be able to load any civilian, an ambulance team must be at the same position as the civilian. That means that the properties 'position' of the civilian and the ambulance team must have the same value.
When a civilian is loaded onto an ambulance, his position become the ID of the ambulance.
An ambulance team can only load one civilian. If it wants to load an other civilian, it must firstly unload the first civilian (who is in the ambulance).

- Content:

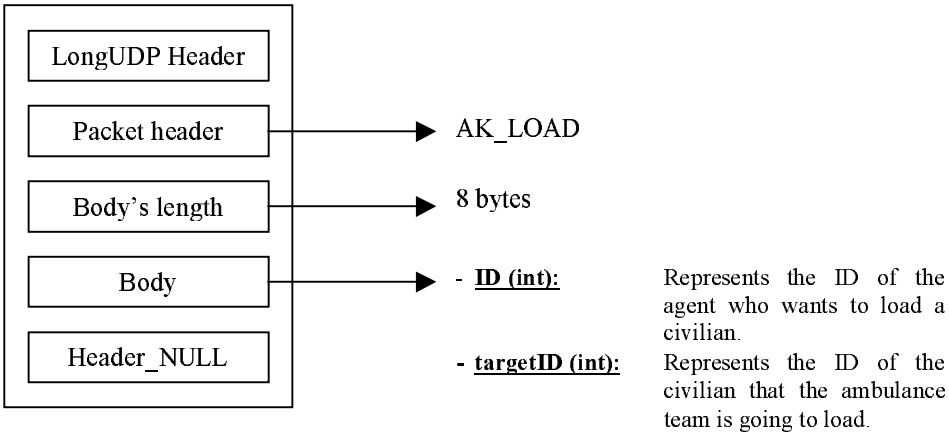


Figure 26: AK_LOAD packet.

8) UNLOAD

- Description: Unload is used to unload an civilian from the ambulance.
- Actors: ambulance teams.
- Comments: Note that no targets is specified. That is because it is of no use: there is at most one civilian in the ambulance. So if the ambulance team uses this command, only one civilian can be unloaded.
- Content:

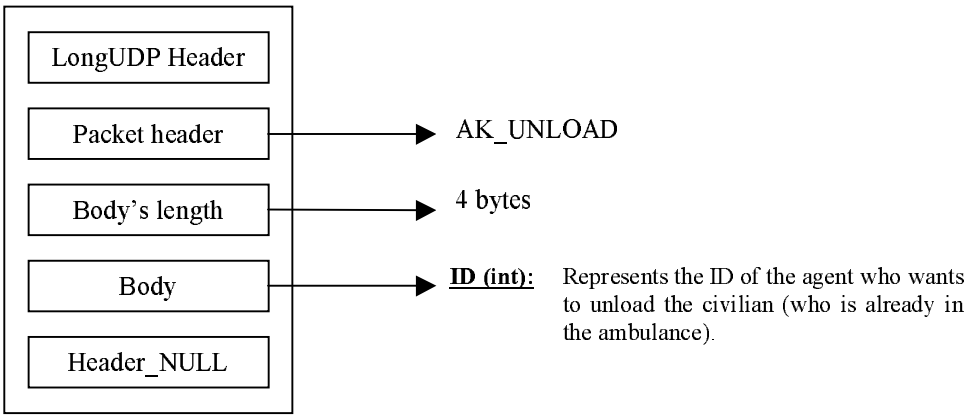


Figure 27: AK_UNLOAD packet.

9) CODE EXAMPLE[18]

This code shows how to create the message to send to the *kernel* to effect an action. In this case, the "extinguish" command is described, the other actions have the same kind of code.

```
VOID EXTINGUISH(INT TARGETID, INT DIRECTION, INT POSITIONX, INT POSITIONY, INT QUANTITY)
        {
        (...)
        CREATE_HEADER_LUDP();
        TABHAUT.WRITEINT(AK_EXTINGUISH);
        BYTE[] BODY = NEW BYTE[4 * 7]; INT I=0;OUTILS.BYTEARRAYCOPY(ID,BODY, I); I+=4;
        OUTILS.BYTEARRAYCOPY(TARGETID,  BODY, I); I+=4;
        OUTILS.BYTEARRAYCOPY(DIRECTION,  BODY, I); I+=4;
        OUTILS.BYTEARRAYCOPY(POSITIONX,  BODY, I); I+=4;
        OUTILS.BYTEARRAYCOPY(POSITIONY,  BODY, I); I+=4;
        OUTILS.BYTEARRAYCOPY(QUANTITY,  BODY, I); I+=4;
        OUTILS.BYTEARRAYCOPY(0,  BODY, I);
        TABHAUT.WRITEINT(BODY.LENGTH);
        TABHAUT.WRITE(BODY, 0, BODY.LENGTH);
        TABHAUT.WRITEINT(HEADER_NULL);
        }
```

7. NEXT *KERNEL*

So, till now, we describe the project at his current state. But RoboCup-Rescue simulation is a new project and needs improvements. At this time, the version of the kernel used is the version 0.39. We worked with the package Ver.0.31 (June 24, 2001) containing all the modules used during a simulation (kernel, GIS, ...). This package represents the competition environment of RoboCup2001.

But now with the version 0.39, which will be the environment of the next competition in June 2002 in Fukuoka (RoboCup2002), there are some little changes for agent developers. The evolution is not major but only one really affect the agents. In previous releases, the kernel ignored the positionExtra property and regarded the middle point of the road as the position of the agent if it is on a road. In the package version 0.39, the kernel regards the true position of the agent.

[18] Outils is a static class which implements some methods of general utility. For more information, please see our codes in the Annexes.

Later, other evolutions are planned to increase the credibility of the Simulation project. Here are mentioned some of them:

- To represent one agent by one object. In fact now a rescue agent represent a set of agents.
- To increase the size of the map.
- To restrict the quantity of water available from a fire hydrant. This is a serious problem to deal with during the real earthquake and so, it is very important to integrate such an issue into the Simulation.
- To develop more realistic civilian.

So, in this chapter, we have described the modules of the project and their probably evolution. Now, we submit some suggestions for the next versions of the RoboCupRescue simulation project:

- To use the centres as the real brain of the cooperation between the agents. That means that centres would be able to receive **all** the messages diffuse by the agents of the same kind and to transmit those they want (not only four). That should allow a better management of the cooperation among the agents.
- To allow safe civilians to help the rescue agents to save others peoples in the debris. This is developing a kind of altruism among the civil population which is a normal behaviour for human as seen on the September 11, 2001.
- To develop an efficient radio system among the rescue agents of the same kind. That means to allow an agent to speak, using telecommunication means, with somebody in particular. This kind of transmission can be compare to the cellular phones.
- To develop the police centre like a radio operator guidance. So, the police centre would be able to inform the population (civilians and rescue agents) of the state of the roads among the map. With this kind of management of the city map, it is possible to inform people of the alternative route (without road blocked) or the deviations available. Police forces would firstly clear the principal axes of the city.

8. CONCLUSION

Knowing those specifications, we had to develop new agents. A good suggestion shall be to develop a low level layer on which an agent layer would come leaning. The first one can be coded in any programming languages and would insure all basics stuffs such as connection, data representation, primary actions and so on. While the other, the high level

layer, would implement all the complex agent's behaviours: their choices, communications and cooperation.

The best thing to do is to code this agent's layer not in an Object-Oriented language as Java or C++ but indeed in an Agent-Oriented language which may contain for example very potent AI algorithms or communication protocol already implemented. This would be a serious help for agents developers but, unfortunately, this kind of programming languages are not yet completely developed. It's possible to find some good prototypes on the Internet such as:

- AgentBuilder (http://www.agentbuilder.com),
- ABE (http://www.networking.ibm.com/iag/iagsoft.htm),
- Grasshopper (http://www.ikv.de/products/grasshopper),
- Knowbot® (http://www.cnri.reston.va.us/home/koe).

V. OUR AGENTS

1. INTRODUCTION

Our first step, as agents developers entering into the research team of the RoboCupRescue Simulation, was to join the mailing list. To do that, we sent an e-mail to the administrator: Ranjit Nair (nair@isi.edu), introducing ourselves in order for the members to understand who we were. From this moment, we received daily news of the international research situation. We could also send our questions, recommendations or answers by writing at the following address: r-resc@isi.edu. Notice that there is a Japanese mailing list: its administrator is Tomoya Nakanishi (nakanisi@r.cs.kobe-u.ac.jp) and its address is: rescue@r.cs.kobe-u.ac.jp.

Once installed, we had to inquire about the simulation project. We went directly to the official Web site of the RoboCupRescue Simulation: http://www.r.cs.kobe-u.ac.jp/RoboCupRescue. There we found all the information we need: the overview of the project (a good introduction for a beginner), the rules for the next competition, several documents and interesting links, the latest news (for those who are not on the mailing list), … After two months of attentive reading about RoboCupRescue Simulation and multi-agent programming, we decided to begin our program.

2. MATERIAL AND SIMULATION

How to get started? Since 2000, all modules of RoboCupRescue Simulation have been implemented on Linux. Games of RoboCup 2001 Rescue Simulation Leagues were played on Linux (RedHat). Using Linux is the quick way to start Rescue Simulation.

The Simulation requires at least three Pentium III 500MHz 256MB Linux for all the simulators and the agents. If your agents need computational power, you should have more machines for agents. But for us, it was a bit different: we had a Pentium IV 1700 MHz 256 MB only for our agents and a biprocessor (2 x 1400 MHz) machine for all the simulators.

The following steps show how to install Rescue Simulation environment:

- Download RedHat from: http://www.redhat.com.
- Prepare Linux machines.
- Download Simulation package from: http://www.r.cs.kobe-u.ac.jp/RoboCupRescue/download.html.
- Unpack files:
  - gzip -d rescue-0***-unix.tar.gz
  - tar xvf rescue-0***-unix.tar
- Follow the README.TXT file.
  - configure
  - make (note: it takes about an hour to compile all files)
- Prepare JDK.
  - *viewer* requires Java environments. If your Linux distribution does not include Java, you need to download it from: http://java.sun.com.

Once all these phases completed, you have installed the environment of the Simulation: *GIS*, *kernel*, *component simulators* and 2D *viewer*. The last thing to do is to implement the agent module or to connect other agents already coded.

Here is some information to help you launch our agent module: download and unzip our codes (and some documentation) from: http://www.info.fundp.ac.be/~eleconte/Code.html. The two packages are already compiled, so you just need to launch the simulation (see the README.TXT file of the Simulation package) and after that, launch our agents using the following command:

JAVA RESCUEAGENT/MAIN *FB AT PF* [*IP* [*PO*]] [19]

The arguments of the command are:

- *FB*, the number of fire fighter brigades. With the version 0 of the *kernel*, this number was fixed at 10, but it can change in the future versions of the Simulation package.

[19] [...] means the argument is optional.

- *AT*, the number of ambulance teams. With the version 0 of the *kernel*, this number was fixed at 5, but it can change in the future versions of the Simulation package.
- *PF*, the number of police forces. With the version 0 of the *kernel*, this number was fixed at 10, but it can change in the future versions of the Simulation package.
- *IP*, this is the IP address of the *kernel*. If it is on the same computer as the agent module, it is not needed to specify this argument.
- *PO*, this is the listening port number of the *kernel*. Normally, the *kernel*'s listening port is 6000 and it is no use to specify it unless the *kernel* changes his port.

3. PROGRAMMATION LANGUAGE

The first decision to take when you decide to code agents is to choose the programmation language you will use. There are not any restrictions in this choice: if you want to use Pascal or Fortran or even Assembler it's possible, even if it seems to be more complicated. All the simulation's modules are written in C or C++. It explains that in the passed competition (Seattle, 2001), most agents modules were developed in C or C++. But some were coded in Java (as the winner: YabAI, http://ne.cs.uec.ac.jp/~morimoto/rescue/yabai).

So, we did not have any constraints. We hesitated a lot between C++ and Java because an object-oriented language seems relevant to implement agents behaviours. We tried to evaluate these two languages: Java is very close to the language C++ given that it has almost the same syntax. However Java is simpler than the language C++ although it is inspired by it, because the critical characteristics of the language C++ (those that are at the origin of the main errors) were abolished:

Java	C++
Predefined types.	Possibility of creation of types.
A single type of structure: the class.	Different types of structures: struct, union, ...
Better management of the exceptions.	
Simple inheritance.	Multiple inheritance.
No object's destructor due to the garbage collector.	Freedom of the memory management (constructor/destructor)

Table 14: Java Vs. C++.

These modifications do not prevent Java from being at least as powerful as C++. However Java is much slower, in fact, it loses in speed what it gains in portability.

We also hesitated to make an API C++/Java for the RoboCupRescue developers, but we decided to forget this possibility because we did not know enough about the Simulation project to appreciate what is more interesting to code in C++ than in Java. So, our decision was to begin to code our agent from scratch and only in Java.

4. METHODOLOGY AND PRESENTATION OF OUR ARCHITECTURE

We had to completely develop the rescue agent (fire fighter brigade, police force, ambulance team and their centre). From the low level connection to the *kernel* to the high level behavioural algorithms.

First of all, we decided the general structure of our programme. We had at this time a good idea of the work we had to do because we read a lot of previous rescue agent programs. We implemented two separate packages to divide the map information (data) and the agent's behaviours (algorithms). The packages' names are:

- ObjectProperties: package which contains all the map objects,
- RescueAgent: package which contains all the algorithms.

The Figure 28 is a representation of our agent module:

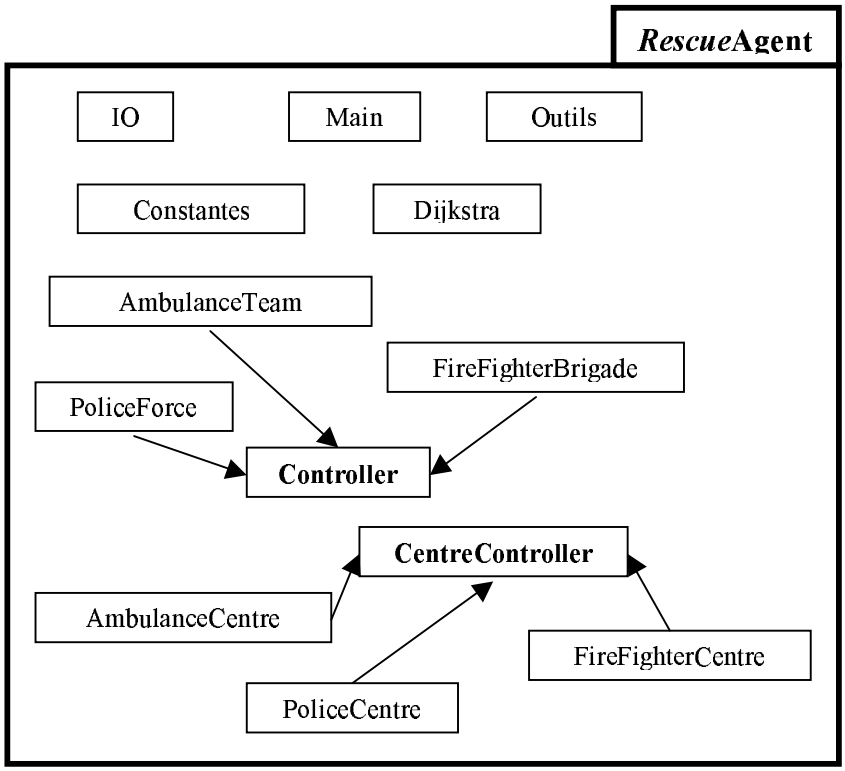


Figure 28: First version of the RescueAgent package architecture.

This is the first version of the RescueAgent package. We spent a lot of time elaborating such an architecture because a good architecture is the best way to begin developing agents.

- Our first goal was to connect us to the *kernel*. We wanted to establish the connection to the *kernel* (using the LongUDP protocol) and to create methods to form the packet for the actions (move, extinguish, clear, say, …). All these methods are in the **IO class**.

- Secondly we planned to create a **Main class** which has to launch the agent simulation. The aim of this class was also to launch the requested number of agents and check the parameters of the server (IP address and port number).

- We also created a tools class which contains a series of useful methods (**Outils class**).

- Then we worked on the rescue agent as an individual just as we would a civilian. We created a common representation of the basic agent (**Controller class**). This class implements some actions as move, say and tell which are common to all the agents.

- The map is a kind of graph with nodes and arcs. So we had to choose and implement a shortest path algorithm. We choose the Dijkstra algorithm and we improved it a little (**Dijkstra class**).

- We had to think about the specific agent: fire fighter brigade, police force and the ambulance team. They extend the Controller because each of them are first of all a simple agent. But every particular agent has some specific actions: the fire fighter brigade's action is to extinguish a fire, the police force's action is to clear the road, the ambulance team's actions are to rescue a civilian, to load the civilian in the ambulance and to unload him from the ambulance (**policeForce**, **ambulanceTeam** and **FireFighterBrigade classes**).

- After that, we had to develop the behavioural algorithms. We thought of creating a special class for all the agent's behaviours. In this way the behaviour can be developed independently of the rest of the agent module (**Behaviour class**). This point is very interesting for the future agent developers to perform the behavioural algorithms.

- To develop the communication among the agents, it is imposed to use the centres. So we developed these algorithms in the following classes: **ambulanceCentre**, **policeCentre**, **FireFighterCentre** and **CentreController** (which is quite the same as the Controller class but for the centres).

Later we added some classes to perform our architecture but we respected the first structure of the RescueAgent package.

5. KEEPING MAP DATA

At the beginning of the simulation, during the connection, all agents receive the map data in the KA_CONNECT_OK packet. During each cycle those data are updated when the agents receive the KA_SENSE packet. This packet holds only the objects containing a property with a changing value.

Each agent has to keep the map data in an individual data structure, it's forbidden to maintain a common map for all the agents (fair play rules). So we need to simulate a kind of personal memory to retain the map information and be able to be updated often. The 25 rescue agents (10 fire fighter brigades, 10 ambulance teams and 5 police forces) need the same kind of data structure to memorise map information. Moreover this data structure has to be powerful, fast and not too heavy.

By using an object-oriented language (Java), each object of the simulated world (a road, a building, ...) can be easily formalised in classes (a road class, a building class, ...). So to be perfectly clear, we create a separate package (ObjectProperties) which contains all objects of the simulated world with their properties.

The Figure 29 shows the structure of the ObjectProperties package:

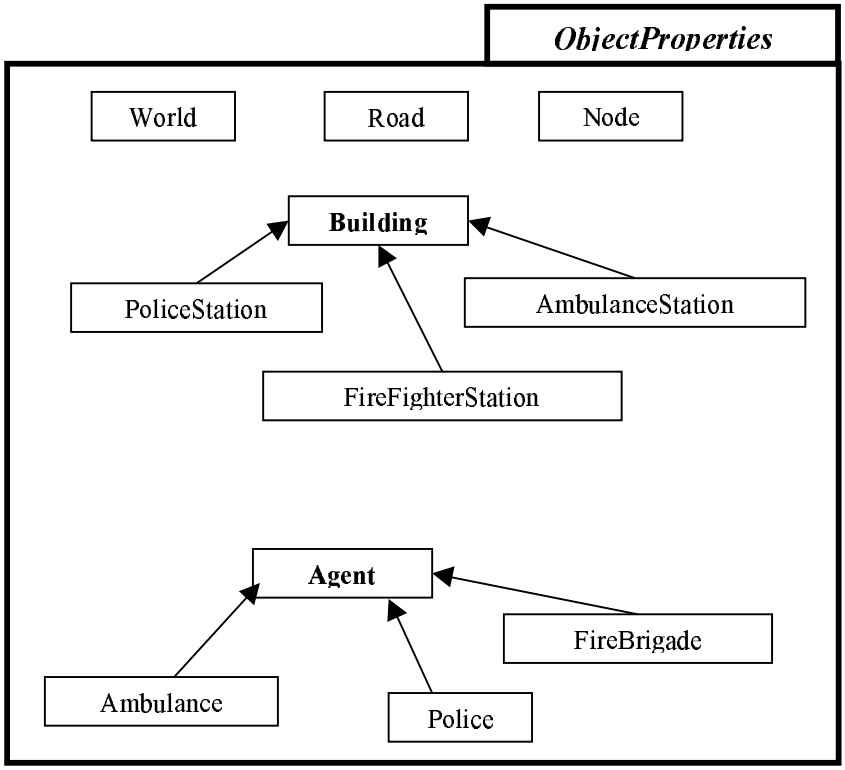


Figure 29: ObjectProperties package architecture.

In the ObjectProperties package, the following objects are developed:

- World: contains all the world's properties (such as the startTime or the windForce).
- Road: contains all the road's properties (such as the length or the roadKind).
- Node: contains all the node's properties (such as the edges or the x coordinate).
- Agent: contains all the agent's properties (such as the position or the stamina).
- Ambulance: contains all the ambulance team's properties. Properties of the ambulance team are the same as

the civilians' ones but for clarity's sake we decide to keep the ambulance class separate.

- FireBrigade: contains all the fire fighter brigade's properties (such as the waterQuantity or the stretchedLength).
- Police: contains all the police force's properties. The properties of the police force are the same as the civilians' ones but for clarity's sake we decide to keep the police class separate.
- Building: contains all the building's properties. Note that the refuge is a building but the property of type is different.
- AmbulanceStation: contains all the ambulance station's properties. The properties of the ambulance station are the same as the buildings' ones but for clarity's sake we decide to keep the AmbulanceStation class separate.
- FireFighterStation: contains all the fire fighter station's properties. The properties of the fire fighter station are the same as the buildings' ones but for clarity's sake we decide to keep the FireFighterStation class separate.
- PoliceStation: contains all the police station's properties. The properties of the police station are the same as the buildings' ones but for clarity sake we decide to keep the PoliceStation class for possible extensions.

Each object of the world contained in the ObjectProperties package has the same format. First of all, the properties of the object are detailed and after that, some methods. The properties are those developed in the previous chapter and are materialised by using different kinds of variables (integer, array, ...).

The methods are generally the same for all the objects:

- PUBLIC VOID SETPROP (INT[] DATA): this method receives an array of integers which contains all the values of the object's properties and it puts the right value to all properties.
- PUBLIC VOID PRINTPROP(): this method prints on the screen all the object's properties values.

All the objects of the simulated world are represented in the ObjectProperties package. But it is not yet sufficient. We have to build a data structure which can conserve all the different objects of the world. We thought a dynamic structure would be more efficient. In our case we have the choice between the vector and the Hashtable. We have chosen to use an Hashtable[20] because there is an more useful index management. We work more exactly with an HashMap. The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronised and permits nulls.

6. TALKING WITH THE *KERNEL*

To be able to exchange information with the *kernel* is the first thing to develop an agent module. So, we begun our program in coding the IO (input/output) class which is, in fact, the lowest level class. This class connect our agent module directly to the *kernel* using the sockets. The *kernel* module may be on the same computer or on an other computer and, in this case, the IO class will use the Internet. Note that through the *kernel*, the agent module is connected to all the other modules, which are also plugged to the *kernel* (Figure 3, page 40).

Now that the agent module and the *kernel* are connected, the communication between them may begin. The IO class insures the bi-directional communication: on one hand, it is charged of the sending of packets from the agents towards the *kernel*. Those packets are sent during the connection of the agent (the CONNECT and ACKNOWLEDGE methods[21]) or when an agent

[20] The Hashtable class from the java.util package allows to create collections of objects associated to a name (a key). An Hashtable can contain different kind of objects.

[21] For more information about these methods and the IO class, please see our codes in the Annexes.

wants to do something (the MOVE, SAY, TELL, EXTINGUISH, ... methods). The *kernel* receives all those messages and takes charge of it: verifying them, executing them or forwarding them to the right module(s).

On the other hand, the IO class warrants the receiving of packets coming from the *kernel* thanks to the RECEIVE_CONNECT method (which only receives the KA_CONNECT_OK packet) and the RECEIVE method (which receives all the packets during the Simulation except the KA_CONNECT_OK packet).

Here finishes our low level layer (connection and data representation). Upon it, we implemented a second layer which contains all the common and basic algorithms of the agents. The main are the moving algorithm, which helps the agent to find the shortest path between two points of the map and to move across it. The other is the communication protocol, this one gives a minimum language to agents. Given that, they are able to coordinate their actions.

7. GENERAL IDEAS

Our agent module contains in fact three kind of agents: the fire fighter brigade, the police force and the ambulance team. To handle all these different agents during the simulation, we use multithreaded programming. So we have a fire fighter brigade class that will be instantiated ten times ; the police force class, ten times too ; the ambulance team five times. So we have 25 threads running simultaneously during the simulation just to represent the rescue agents. We cannot develop 25 different behaviours, so we have to think about generic algorithm for each kind of rescue agents.

It's also necessary to think about the way the agents will react to the sensory information they receive. But all the agents will not have the same behavioural algorithms: the fire fighter brigades have to react when they see or ear fire and they have to try to extinguish it. The police forces have to clear a road when they see or ear that a road is blocked. And finally, the ambulance teams have to save (rescue, load in the ambulance and unload) civilians who need help.

In fact the rescue agents have to react immediately to changes in the environment. We do not think now about the way the agent has to react to the environmental update, we just try to imagine how the agent can be aware of their environment and react simultaneously. The solution was to use a

thread to react to the sensory information the agent receives each turn. We call this thread the reaction thread. Even if this thread is different for each kind of rescue agents, the principle is quite the same.

The Figure 30, here after, shows the way the rescue agent reacts when something interesting for him happens. It will be explain in more detail after the following figure.

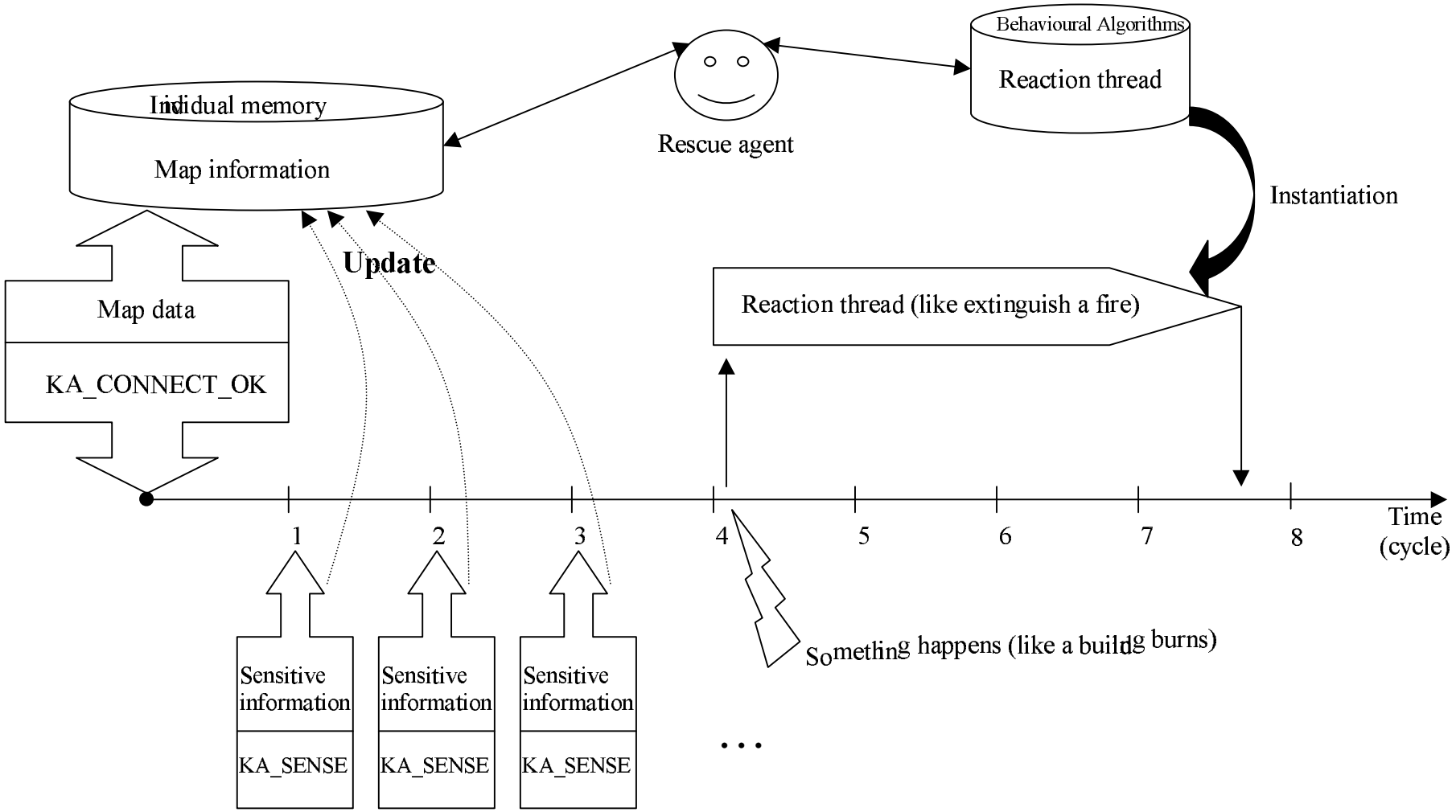


Figure 30: Reaction thread.

This figure details the reaction scheme of all the agents. Here follows the description of this reaction:

- On one hand, the rescue agent has an individual memory containing all the map information and on the other hand, it has his own behavioural programme. The behavioural algorithms contain, of course, the reaction thread.
- The agent receives the map information during the connection (thanks to the KA_CONNECT_OK packet) and each cycle it receives the updates of the map in the KA_SENSE packet.
- Until now everything is all right, no special sensory information is noticed. That means no fire, no block road or no civilian who need help.
- Suddenly something happens (just at the beginning of the fifth cycle): the event is a fire or debris blocking a road or a civilian who needs help.
- Then the behavioural algorithm starts and decides to react or not. Sometimes it will be more efficient to do nothing and to wait for another event than to react immediately. But this is a part of the specific behavioural algorithm explained below.
- If the agent reacts, the behavioural algorithm launches the reaction thread. This thread consists of saving a civilian for a ambulance team, extinguishing a fire for a fire fighter brigade and clearing a road for a police force.
- At this time the agent thread continues to run normally and be aware of the environmental information. Simultaneously, the reaction thread continues its way.
- When the reaction thread finishes, that means the civilian is saved or dead, the road is clear or the building does not burn anymore, the reaction thread kills itself, but the agent thread is still active.

Using another thread is very useful because it's necessary to continuously update the map information to know if the situation changes or not. In fact, just imagine that we do not use another thread. The agent should only react or be aware, but not both simultaneously. So if something happens, the agent should react and then cannot be aware all the time to the new sensitive information. It might be able to manage a system of this kind but this would surely be less efficient and moreover less realistic. The reaction thread allows the agent to be aware all the time of the world's information and to react at the same time.

Concerning the concurrent accesses, we have two threads: the "agent thread", the one which update the map information, and the "reaction thread". The first one read and, of course, write on the map information while the other thread is only reading it. So, it is not possible to have any

writing problem. Concerning the reading, it is not a real problem because the "reaction thread" constantly (each cycle) read the map information.

The reaction thread is the general idea of implementation for our three kinds of agent. However each kind of agent applies this idea in a different way. But these personal applications of the rescue team are explained in the sections 10, 11 and 12 of this chapter.

8. MOVING ALGORITHM

1) INTRODUCTION

In the simulated world, agents have to move across the map using the special action called *move*. This action has been detailed in the previous chapter and needs parameters. One of those parameters is the way taken to move from the start point to the arrival point. So we have to think about a suitable way of moving across the map. This way has to be the shortest way or the less collapsed (with the smallest number of blocked roads).

But the second solution is impossible due to the fair play rule (the individual memory of the agent). An agent cannot know, for all the roads of the map, if this road is blocked or not. So, we looked for a shortest path algorithm in the simulated map. But before thinking about a complex algorithm, it's essential to formalise the data we have: the map.

2) THE SIMULATED WORLD MAP = GRAPH

Each agent receives the map at the beginning of the simulation and updates some objects of this map each cycle. The map contains roads, buildings and nodes. Intuitively, the map can be assimilated to a graph.

Effectively, a road could be represented by an **arc**[22] of a graph. The direction of the road could be shown in a graph. If the road has a double direction then there is no arrow on the graph. A **node** of the graph could represent a building or more generally a node of the map.

[22] In bold: the mathematical words used ; to avoid a confusion between the node of the map and the node of the graph.

Graphically, the objects of the simulated world evaluated are:

Simulation object	Graphical object	Mathematical object
Building and node	**Node**	x_i
Road	Arc	$u_k = (x_i , x_j)$ with $i \neq j$ and $\forall k, \neg \exists r: u_r = (x_i , x_j)$.
Length of the road	weight of the vertex	$p(u_k) = p(x_i , x_j)$

Table 15: Graph data.

We notice that the length of a road is always a positive value whatever the direction used to pass through it.

Mathematically:
- $U = \{ u_k \}$, the set of all the roads of the map.
- $X = \{ x_j \}$, the set of all the **nodes** of the map.
- $\forall k \in U ; p(u_k) \geq 0$, the length of all roads are positive or equal to zero.

3) THE PROBLEM

Now that the data is formalised, we can set-out the problem. The most important thing for a rescue team in an earthquake is the time management. So it seems very important for rescue agents to move quickly across the map. Logically, the quickest way is the shortest one. However, there are a lot of blocked roads after an earthquake and it should be very useful to avoid those roads if possible. But, as said before, an agent does not have all the information about all the blocked roads of the map. So, the only possibility is to find the shortest path and to ask the help of a police force when it's necessary.

An other way of thinking would lead us to the following idea: if an agent comes up against a blocked road, it tries again to find the shortest path till its arrival point avoiding this blocked road. But again, we disregard this possibility. In fact, just after an enormous earthquake as the Kobe's one, maybe 70% of the roads are blocked. So we risk, at the beginning of the Simulation, that the agent do not find any road towards their destination. But this improvement of the agent's moving algorithm could be powerful after 50 or 75 turns, when police forces have already clear many roads.

So, our problem is to find the best path from a node, a building or a road (**node** or **arc**[23]) to another node or building (**node**). The best path is the shortest one or the lighter one in our case. This problem is a classic of graph theory. A lot of mathematicians try to solve it and there are a lot of different solutions possible. We choose the Moore-Dijkstra algorithm which seems to be the most powerful.

4) OUR SOLUTION: DIJKSTRA'S ALGORTIHM

We use the same mathematical conventions developed in the part 2 of this chapter.

We have to pose:

$p(x_i, x_j) = 0$ if $x_i = x_j$

$= +\infty$ if $(x_i, x_j) \notin U$

Suppose that we have a graph with n **nodes**. The algorithm runs in n-1 iterations. The algorithm updates an integer array: pi. The pi array has a length of the numbers of nodes and each item of the array corresponds to a node. So the pi[i] item corresponds to the node x_i.

The algorithm uses another set, D. In fact, the X set is divided into two parts: D and X \ D.

- D contains the **nodes** already visited (for which the shortest path is already calculated).
- X \ D contains the other **nodes**.

At the beginning of the algorithm, we must update the pi array with the value of the start point. In fact the pi array contains all the weights from the start point to the other **nodes**. So the $pi[x_r]$ contains the weight of the way from the start point to the x_r node. So the Dijkstra algorithm consists of filling the pi array with the lightest weight for each **node** of the graph. This principle of updating the pi array is called the relaxation principle.

Suppose that x_0 is the start point:

- the value of $pi[x_0]$ is 0;
- for each other **node** of the graph which has a connection with the start **node** we have to update its value of the pi array with the value of the vertex (the length in our case) between the two connected **nodes**. (relaxation principle).

[23] In version 0.31, an agent cannot stop itself in the middle of a road, but when the road is blocked it can pass through a certain part of the road and then be blocked somewhere else on the road.

After that, each iteration happens this way:

- choose the **node** not already visited with the lighter value in the pi array,
- for each other **node** of the graph which has a connection with the selected **node,** we have to update his value in the pi array. The new value is the minimum value between the weight of the vertex between the two connected **nodes** added to the value of the selected node in the pi array and his previous value (relaxation principle),
- consider that the chosen node is now already visited.

Algorithmically we can formalise Dijkstra's algorithm this way (suppose that x_0 is the start point):

$D = \{ x_0 \}$
$pi(x_0) = 0$

for $x \in X \setminus \{ x_0 \}$ do:
  $pi(x) = p(x_0, x)$

While $D \neq X$ repeat:
  choose $y \in X \setminus D$ such that $pi(y) = \min \{ pi(x) \mid x \in X \setminus D \}$
  $D = D \cup \{ y \}$
  $\forall x \in X \setminus D$ do:
  $pi(x) = \min [pi(x), pi(y) + p(y, x)]$

5) IMPLEMENTATION

To implement Dijkstra's algorithm we create two classes (included in the RescueAgent package): Dijkstra.java and WeightBefore.java. Dijkstra.java contains the algorithms and WeightBefore.java is a data structure. Here follow a description of our implementation.

First of all Dijkstra.java contains data:

```
PUBLIC CLASS DIJKSTRA IMPLEMENTS CONSTANTES
    {
    /** CONTAINS ALL THE OBJECTS OF THE MAP.*/
    PUBLIC DATA MYDATA;
    /** CONTAINS THE ID OF START NODE OF THE ALGORITHM.*/
    PUBLIC INT START;
    /** CONTAINS THE ID OF THE FINISH NODE OF THE ALGORITHM.*/
    PUBLIC INT FINISH;
    /** IS THE DATA STURCTURE USE TO RETAIN THE WEIGHT AND THE PREDECESSOR OF EACH NODE.*/
    PUBLIC HASHTABLE DIJ_DATA ;
    /** CONTAINS THE BRAND NEW VISITED NODES AF DIJ_DATA (PERFOMING DIJKSTRA).*/
    PUBLIC VECTOR JUST_VISITED;
    (…)
    }
```

The pi array is represented by dij_data in the code. Dij_data has a particular structure: it's an hashtable of WeightBefore. The WeightBefore has the following fields:

```
PUBLIC CLASS WEIGHTBEFORE
    {
    /** CONTAINS THE WEIGHT OF THE SHORTEST PATH FROM X0 TO THE OBJECT.*/
    PUBLIC INT MYWEIGHT=0;
    /** CONTAINS THE PREDECESSOR OF THE OBJECT ON THIS SHORTEST PATH.*/
    PUBLIC INT MYBEFORE=0;
    /** CONTAINS THE ID OF ROAD BETWEEN THE PREDECESSOR AND THE OBJECT ON THIS SHORTEST
    PATH.*/
    PUBLIC INT ROADBEFORE=0;
    /** IF VISITED IS TRUE, THAT MEANS THAT THE OBJECT BELONG TO D AND THAT THE VALUES ARE
    FINAL.*/
    PUBLIC  BOOLEAN VISITED=FALSE;
    (...)
    }
```

This structure is essential to retain the values of the weight (*MyWeight*) and predecessor (*MyBefore*) necessary for Dijkstra's algorithm. The predecessor is useful to compute the path.

We add a vector to dij_data to improve the result: *just_visited* which contains the ID of the most recent update values from dij_data. This is useful to perform the time asked for the resolution of the algorithm. In fact, in its original version (that means without any change of the classical Dijkstra's algorithm), it takes too much time to be executed (complexity of $O(n^2)$). By introducing *just_visited* we reduce the number of **nodes** visited (by forcing the visit of the **nodes** with most recent changing values), so we decrease the resolution's time.

We provide each agent with its Dijkstra object to allow more simultaneous moves. So with our 25 rescue agents we have 25 Dijkstra's. An agent use all the simulation time the same Dijkstra for each move across the map. So to implement a reusable object we imagine two main methods: the initialisation (*set_data*) and the solving of the algorithm with a start and finish points (*solve*).
The first method, *set_data*, create dij_data and is only used once by each agent. It fills the dij_data with the agent's particular data's. The *set_data* method is called once at the begin of the controller execution of each agent.

The second method, *solve*, is used each time the agent needs a path between two points (at each movement so). *solve* updates the dij_data object with the values of the start and finish points (by relaxation) and, by this way, calculate the way between the two points. So, first of all, there is an initialisation of the dij_data with the values of the start and the finish points.

Concretely we use a method called *init*. Then, the resolution by using relaxation process, we progress by filling of the dij_data structure. We change something in the resolution in front of the classical Dijkstra: when there is a way found the algorithm stops immediately, to reduce the time of the execution.

9. COMMUNICATION PROTOCOL

The RoboCupRescue Simulation allows the agents to speak together. If it is used judiciously, it can seriously improve the reaction of the rescue agents. There are not any language specification, so you are totally free to create your own language. We just know that the civilians use the FIPA ACL[24] language to call up for help.

So, any agent can communicate with other agents. When an agent *says* a message using his voice, the message is immediately heard by other agents within a radius of 30 meters. When an agent *tells* a message by communications devices, this message is immediately heard by agents whose type is the same as the speaker's, and his centre. Furthermore, when a centre agent tells something, the message is transferred to other type centres (Figure 31).

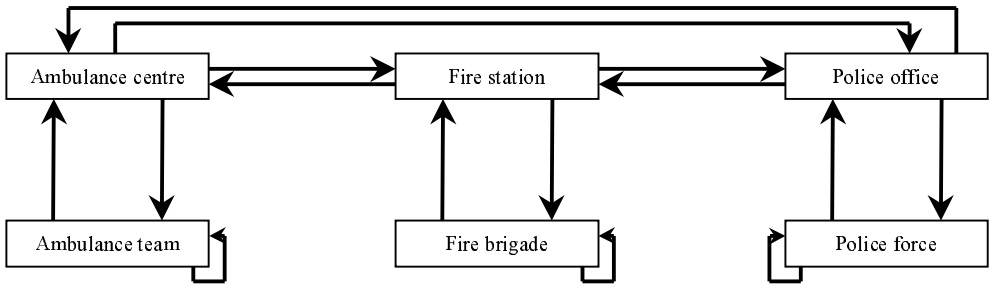


Figure 31: Tell by communication devices.

But to be more realistic, agents can only receive (and send) four messages during one simulation turn[25]. Unfortunately, centres are considered as

[24] More information about it on: http://www.fipa.org

[25] The 300 turns of the simulation represents the first five hours after the earthquake, so one turn is equivalent to one minute.

normal agents and have the same restrictions which seems to be inadequate for them. So we have to minimise the number of sentences said (and received) by an agent.
To limit the receiving, when an agent receives hear information from the simulation system, the agent may select whether it hears the message or not by checking who is the speaker.

For the sending limitation, our agents can only send tree type of messages:

- ROAD_BLOCKED

This message is sent by an agent to call up for the help of a police force to clear a route segment. Of course, only fire fighter brigades and ambulance teams will send this kind of message (to police forces).

The syntax of the message is:

"ROAD_BLOKED"	ROAD.ID	ARRIVAL.ID	ME.ID

with "ROAD_BLOKED", string which identifies the kind of the message.
ROAD.ID, identifier of the blocked road (start point).
ARRIVAL.ID, identifier of the route's point[26] the agent wants to reach (arrival point).
ME.ID, identifier of the agents who send the message.

- BUILD_BURN

This message is sent by an agent to warn the fire fighter that a building is on fire. Of course, only police forces and ambulance teams will send this kind of message (to fire fighter brigades).

The syntax of the message is:

"BUILD_BURN"	BUILD.ID

with "BUILD_BURN", string which identifies the kind of the message.
BUILD.ID, identifier of the burning building.

[26] This will only be the identifier of a node or a building.

- NO_NEED_TO_HELP

This message is only send by an ambulance team to other ambulance teams. They send this message when they load a civilian or when they see a dead civilian. The meaning of this message is that the civilian does not need any more help because he is dead (it's no use trying to save him anymore) or he is saved.

The syntax is:

"NO_NEED_TO_HELP"	AGENT.ID

with "NO_NEED_TO_HELP", string which identifies the kind of the message.
AGENT.ID, identifier of the agent (civilian) who is dead or saved.

Notice that all those messages are only sent using the *tell* method. Our agents do not use the *say* method because it is very difficult to know who is in a radius of thirty metres (who can hear the message).

Here follows the highest level layer, the "agent" layer. The two previous layers implement some basic algorithms and primary behaviours. But the next layer will be the most important, and the most difficult for agent developers because it contains all the specific agent's behaviours.

10. FIRE FIGHTER BRIGADES ALGORITHMS

We have ten fire fighter agents in the simulated world. Each represent a team of firemen. The fire fighter brigades are the only agents able to use the "extinguish" command. Moreover, it's much more efficient to work in team to face with a burning building. So it's imperative to develop some cooperation between the fire fighter agents themselves and between the fire fighters and the other rescue agents.

The fire simulator, an element of the *component simulators*, manages the spreading of the fire around the map. At the begin of the simulation, it decides some ignition points which are placed where the fire occurs. At this time there are four ignition points across the map. The fire fighter brigades are aware of this, in fact, just imagine that you can see some gray fogs on the sky when there is an earthquake. These are the first information that the fire fighter brigades have over the burning buildings. During the simulation

they detain other sources of information's to know where are the fires: the KA_SENSE and the "BUILD_BURN" messages from the other agents. So the fire fighter brigades are conscious of the different places where there is fire. Because the fire spreads quickly, it's necessary to react immediately and to conceive some cooperation algorithms.

The first idea we developed is to split the ten fire fighter brigades into the number of ignition points. So with four ignition points we have four teams: two with two brigades and the others with three. This is easy to implement by a simple integer division operation. Then each team is redirected to an ignition point (the nearest one). So from the begin of the simulation each fire fighter brigade has something to do: to move near his ignition point. Possibly during the journey the fireman can extinguish another burning buildings. This is the first step of the fire fighter brigade algorithm which ends after the fifty first cycles or when the ignition point is completely burnt or extinguished.

Secondly the idea is the following: to extinguish the nearest burning building. That means first of all the fire fighter brigades are aware to the information they receive in the KA_SENSE packet and then only if there is no burning building thirty meter around, they hear the messages from the other rescue agents. When a fire fighter brigade has fixed goal as explained in the two paragraphs above, it starts its reaction thread. The reaction thread developed for the fire fighter brigade is quite simple:

If not *busy*, become *busy*, else no reaction.
Extinguish the building while not totally burned or extinguished.
Become not *busy*.

This version of the reaction thread is effective only when the fire stays around thirty meters around. That means when it's not necessary to move for extinguishing. The following version is more realistic and consider the move across the map:

If not *busy*, become *busy*, else no reaction.
Move near the goal if necessary ; that means near the burning building selected as those to extinguish(it may request the help of a police force to clear the road: sending of BLOCKED_ROAD messages).
Extinguish the building while not totally burnt or extinguished.
Become not *busy*.

We can resume the situation:

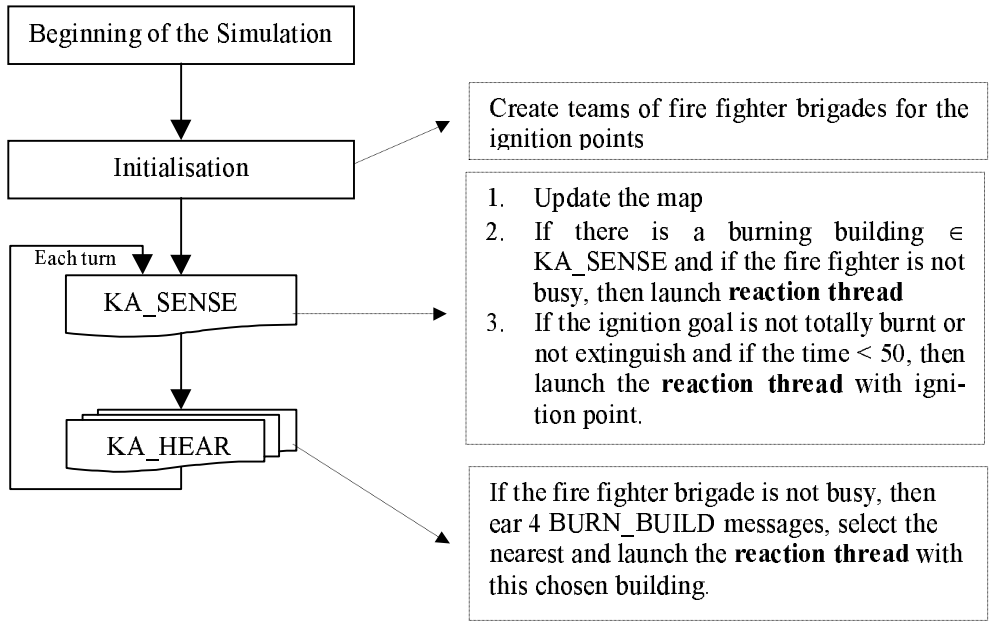


Figure 32: Fire fighter algorithm.

11. AMBULANCE TEAMS ALGORITHMS

The aim of the ambulance teams is to save civilians. To do that, the ambulance teams have to find civilians, to rescue him (to free them from the collapses), to load them and to unload them into the refuge (a kind of make-shift hospital) where they can have some medical assistance. That's the only way an ambulance team can help civilians. Note that they can also save other agents (fire fighter brigade, police force and another ambulance team) which are buried.

Ambulance teams have two information sources to find civilians. The first one is their calls for help. But, unfortunately for the realism of the Simulation, we do not use this information. In fact, civilians send periodically a message which can only be heard by agents who are in a radius of thirty meters. So, if the ambulance teams base their saving plan exclusively on these messages, only few civilians would be saved.

The second information source is the position on the map of all the civilians at the beginning of the Simulation. This is not very realistic, but we have this information (into the KA_CONNECT_OK), so we use it. Our point of

view is to save the nearest civilian. To handle that, the ambulance teams have a data structure to remember which civilians are saved, dead or still buried. The Figure 33 represents the ambulance team behavioural algorithm.

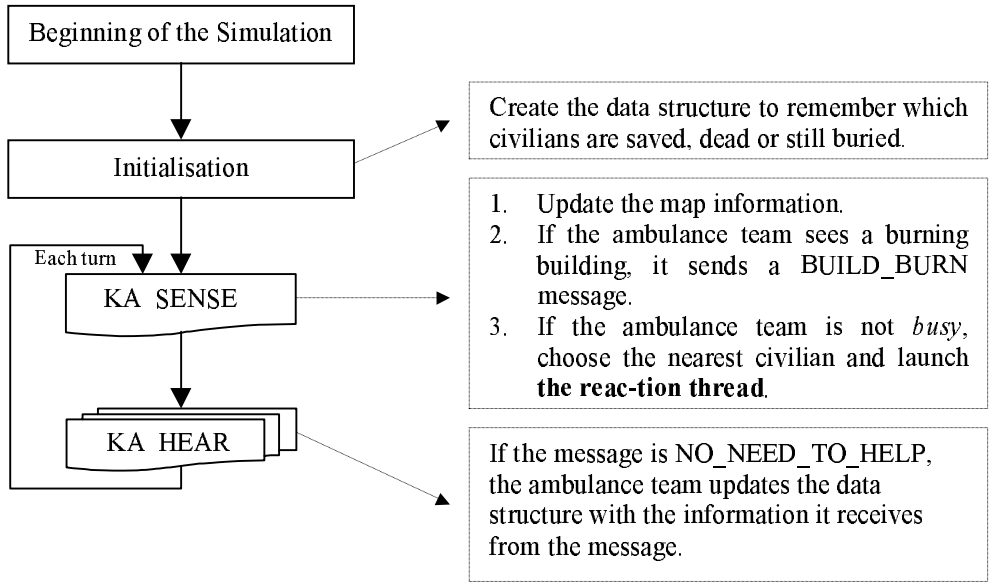


Figure 33: Ambulance team algorithm.

Here is a pseudo code of the ambulance teams reaction thread:

```
If not busy, become busy, else no reaction.

Move to the building where the nearest civilian is to save (it may request the help of a
police force to clear the road: sending of BLOCKED_ROAD messages).

Rescue him (check if it is useful to save him and if not, send a NO_NEED_TO_HELP
message (finish)).

Load him.

Move to refuge.

Unload him.

Become not busy.
```

But when is it useless to save a civilian? There are two reasons to take such a decision: the first one is when the building, where the civilian is, is burning. In this case, it is almost certain that the civilian will die and there is an great risk of losing the ambulance team. Given that, our ambulance teams

do not try to save civilians when they are in burning buildings. The other reason which could persuade us to abandon a civilian is when the property hp of the civilian is too low. In fact, civilians have three important properties for this calculation: hp, damage and buriedness (Table 8, page 63). Each cycle, the miscsimulator subtracts the value of damage from the value of hp. So, while the civilian is not saved (buriedness=0):

$$hp_{t+1} = hp_t - damage$$ [27]

So it's interesting to save a civilian only when:

$$hp - (damage * buriedness) > 500$$ [28]

12. POLICE FORCES ALGORITHMS

We divide the 10 police forces into two teams of 5 police forces each. The first team, composed of five police forces, only listens to the messages coming from the five ambulance teams. So we assign one ambulance team to one police force and during the simulation this police force will listen to the messages coming from this ambulance team only. The other team, composed also of five police forces, listens to the fire fighter brigades. We assign one police force to one ignition point. So, during the simulation, this police force will only listen to the messages coming from the fire fighter brigades assigned to the same ignition point. The Figure 34 represents police force behavioural algorithm.

[27] hp_t represents the value of the property hp at the turn t.

[28] 500 is the approximate hp value needed to reach the refuge.

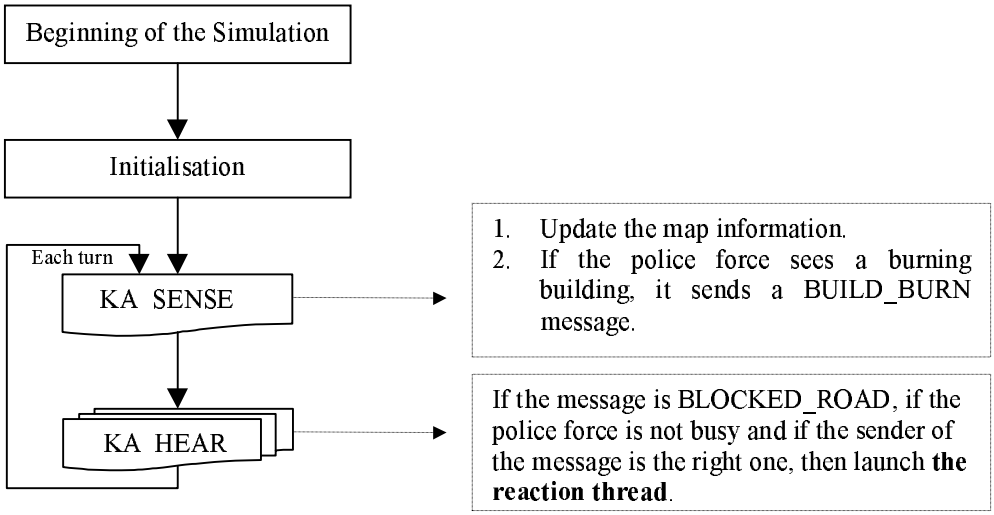


Figure 34: Police force algorithm.

Here is a pseudo code of the police forces reaction thread:

```
Become busy.

Clear the roads from the police force's position to the road with the identifier road.ID.

Clear the road with the identifier road.ID.

Clear the roads from the road with the identifier road.ID to the node (or the building)
with the identifier arrival.ID.

Become not busy.
```

13. CENTRE ALGORITHMS

The only thing a centre can do is to manage the messages it receives. How do they do that? First of all, here is an example to understand the problem: simple situation, a fire fighter brigade calls for the help of a police force to clear a given road (Figure 35).

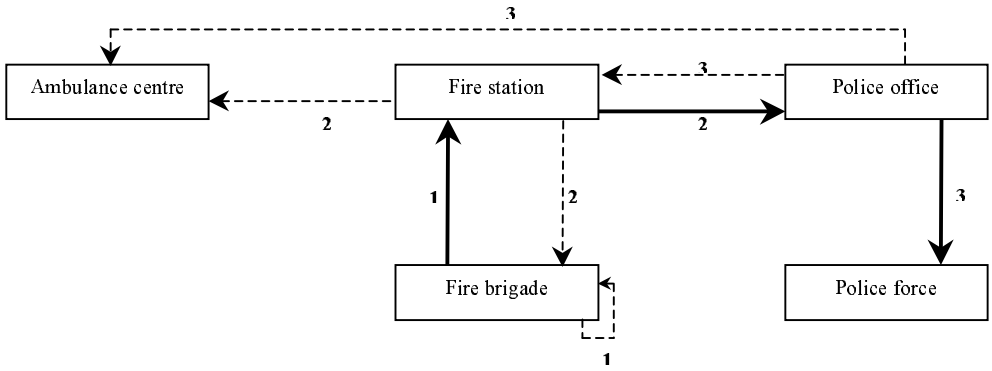


Figure 35: Communication example.

1: The fire fighter brigade *tells* the following message, "BLOCKED_ROAD" + X + Y + Z [29]. As said above, this message is heard by all the fire fighter brigades (who, in fact, do not care about the message) and the fire fighter station.

2: The fire fighter station will forward this message, because it should be heard by the police office. So, the station *tells* again this message which will be heard by the two other centres: the police office and the ambulance centre.

3: Only the police office will take care of the message, and tell it again to all the police forces.

With that simple example, we can see that a fire fighter who sends a message to call a police force produces, in reality, three important messages (bolded lines in the Figure 35): one from the fire fighter brigade to the fire fighter station, an other from the fire fighter station to the police office, and the third one from the police office to the police forces. But it also creates a lot of totally unnecessary other messages (dotted lines in the Figure 35). It's why we have to minimise the number of sentences forwarded by the centres. In order to do that, we made three different algorithms, one for each centre:

[29] The arguments of the message are useless for the example.

- Police Office
  - ROAD_BLOCKED:
    When the police office receives a "ROAD_BLOCK" message, it always forwards it (so that all the police forces receive the message).

  - BUILD_BURN:
    When the police office receives a "BUILD_BURN" message, first of all it checks if the sender of the message is a police force.
    If it is, then the police office forwards the message (so that the Fire fighter station receives it). And if the sender is not a police force, the police office will forgets this message (it is not useful to forward it).

  - NO_NEED_TO_HELP:
    When the police office receives a "NO_NEED_TO_HELP" message, it always discards it (this kind of message is only interesting for ambulance teams).

- Fire Fighter Station
  - ROAD_BLOCKED:
    When the Fire fighter station receives a "ROAD_BLOCK" message, first of all it checks if the sender of the message is a Fire fighter brigade.
    If it is, then the Fire fighter station forwards the message (so that the police office receives it). And if the sender is not a Fire fighter brigade, the Fire fighter station discards this message (it is not useful to forward it).

  - BUILD_BURN:
    When the Fire fighter station receives a "BUILD_BURN" message, it always forwards it (so that all the Fire fighter brigades receive the message).

  - NO_NEED_TO_HELP:
    When the Fire fighter station receives a "NO_NEED_TO_HELP" message, it always discards it (this kind of message is only of use for ambulance teams).

- Ambulance Centre
  - ROAD_BLOCKED:
    When the ambulance centre receives a "ROAD_BLOCK" message, first of all it checks if the sender of the message is an ambulance team.
    If it is, then the ambulance centre forwards the message (so that the police office receives it). And if the sender is not an ambulance team, the ambulance centre discards this message (it is not useful to forward it).

  - BUILD_BURN:
    When the ambulance centre receives a "BUILD_BURN" message, first of all it checks if the sender of the message is a ambulance team. If it is, then the ambulance centre forwards the message (so that the Fire fighter station receives it). And if the sender is not an ambulance team, the ambulance centre discards this message (it is not useful to forward it).

  - NO_NEED_TO_HELP:
    When the ambulance centre receives a "NO_NEED_TO_HELP" message, it always discards it. All the ambulance teams have already heard this message because it comes from an other ambulance team.

14. IMPROVEMENTS

It is, of course, possible to seriously ameliorate our behavioural algorithms. Here is a series of critical points to improve. First of all, the fire fighter brigades could better take charge of the fire. Fire rapidly changes its condition, so it must be especially coped with. It may not be difficult to extinguish an early fire for even a few fire brigade agents, but, on the contrary, it is very difficult to extinguish a late or big fire even for many. In our program, the fire brigades extinguish the first fire they see, but with a little more cooperation, it would be possible to barricade a spread fire by extinguishing edges.

Our agents have also to take charge of the roads. The police forces clears the roads when they receive a "BLOCKED_ROAD" message. But, in order to save others' time, it would be effective to clear important road in advance such as roads leading to refuges and ignition points.

A better management of the messages would be a good progress for our program. Almost each turn, centres receive more than the four messages they can hear, so reduce even more the messages sent by agents and forwarded by centres is the key for a good communication and cooperation among the agents.

Finally, the moving algorithm is, for the moment, very potent. But when the map will get bigger, it will take a lot of time to find the shortest way between two points on this map. That's because the algorithm considers all the roads of the card when it looks for a route. A good idea to decrease the calculation time is to introduce a series of heuristics. The blind research algorithms does not exploit any information concerning the structure of the research tree or the potential presence of solution-node to optimise the research. This is a "rustic" research through the space until finding a solution. Unfortunately, most of the real problems may provoke an explosion of the possible states number. An heuristic research algorithm uses the available information to process more effectively the research. There are many heuristic algorithms already developed such as A* or the Best-First search algorithm which is a combination between research in depth and in width. It investigates nodes in the (increasing/lessening) order of their heuristic values.

15. CONCLUSION

When all the modules are connected, the Simulation begins with the 100 agents: 72 civilians, 10 fire brigades, 10 police forces, 5 ambulance teams and 3 centres. The aim of the game is to save as many civilians as possible without losing a rescue agent. Our agents are able to save between 45 and 55 civilians, that means that we only lose between 17 and 27 on the 97 human agents (centres can not die).

This score is not as good as YabAI's one which saves 61 civilians, but we did not have the same objective. In 4 months, it was not possible to fully develop powerful agents. We did not have enough time to deepen our algorithms, so we tried to go further into something else which misses also in the RoboCupRescue: specifications. Indeed, there was a real lack of specification for new agent's developers, we passed two months to understand how the Simulation goes and how to connect our agent module to the *kernel*. So, we decided to spend a lot of our time to code a clear, flexible and understandable low level layer which can easily be reused by developers of new agent.

Upon this brand new layer, we decided to create an intermediate layer to give some tools to developers who will use our program such as the moving algorithm or the communication protocol.

Finally, only to test these two previous layers, we developed a lighter high level layer which gathers all our behavioural algorithms. This layer is not very effective, but it was not our main objective. Now, we hope other agent's developers will use our lower level layers to develop an efficient and competitive behavioural layer.

VI. CONCLUSION

We must note the activity and the ambition of the industrialists and laboratories of the United States, Japan and certain European countries working in robotics and the Artificial Intelligence. Why such a passion? What are the real reasons which animate these projects? It is impossible to answer precisely to this interrogations because each research project is justified by its own goals. However, would it be at a scientific, computer science or whether psychological level, we can not deny that these projects provide many contributions of knowledge. More precisely, if we are lingering on the project which now occupies us since one year, we are absolutely certain that, in the long run, the RoboCupRescue project will bring to Science invaluable knowledge.

Indeed, RoboCupRescue is a relatively young research project, only two years old. Currently, the project looks like a great play where teams of developers are in competition. Nevertheless, under its appearances of entertainment for big children, RoboCupRescue is a sizeable project with future. Its long-term goal (for 2050) is the following: human technology will have to be able to set up teams of robots which will save lives in real situation and during a catastrophe of great extent. Demonstrations as RoboCupRescue are perfectly studied to make known, in a ludic way, researches which have furthermore a strategic interest.

RoboCupRescue is an ambitious project which deserves to appear among the most significant projects of the Artificial Intelligence. Indeed, it envisages the development at a human size of a time critical organisation of the helps at the time of earthquakes. It is a social project: its goals are to save lives, to limit the casualties and to minimise the material damages. It is also an universal project: the natural disasters break in all the areas of the world, so it unifies various nations around a joint project. It is even a high level research project for computer science. It is actually necessary to develop high level programs with a great algorithmic complexity to simulate reality. Lastly, it requires high level scientific knowledge in the fields of seismology, the geography, of the human behaviour.

So, the principal intention of the RoboCupRescue project is to promote research and development in this socially significant domain at various levels. That involve series of other research project such as: multi-agent team work coordination, physical robotic agents for search and rescue, information infrastructures, personal digital assistants (PDA), standard simulator and decision support systems, human behaviour simulator,

evaluation benchmarks for rescue strategies and robotic systems that are all integrated into a comprehensive systems in future. So, even if RoboCupRescue is quite young, it has many ambitions. Due to that, the project encounter difficulties which will force more researches. For example, real world problems are too complex and too heavy to be assumed by the actual simulator. All the more our knowledge in domains such as seismology , human behaviour or well group psychology remains limited.

Our first objective, when we were integrated in the RoboCupRescue project, was to create competitive agents for the future competition (Fukuoka, June 2002). The people who framed us strongly advised to conceive new agents completely, without drawing one's inspiration from agents already developed. The agents we developed are not of a high level, they obey to algorithms of simple behaviours, but we did not have the time to deepen the behavioural algorithms, or to add an additional layer: an agent layer. However, since the first weeks we worked on the project, we modified our goal in front of the difficulties encountered to collect the necessary information for the development of agents in the RoboCupRescue. Granting more time to formalisation of the agent development, we decided to provide a totally new work.

To this ends, we exposed, within context of this thesis, an theoretical approach as much as practical of the agents development in the RoboCupRescue. On the basis of the theory and leading to practical example of our design of first-aid agents, we have the ambition to add something to the project. Concretely, we have develop a practical handbook for the development of rescue agents. We have specified and formalised as well as possible the design of first-aid workers. This should allow the future developers to integrate more easily the project.

Hoping that our work helps the future generations of rescue agent developers in the RoboCupRescue project!

VII. BIBLIOGRAPHICAL REFERENCES

[AGA68] AGARD, J. (1968) *Les méthodes de simulation.* Dunod Paris.

[ATT97] Attoui, A. (1997) *Les systèmes multi-agents et le temps réel.* Eyrolles Paris.

[BOI01] Boissier, O. (2001) *Systèmes Multi-Agents, Environnement.* Cours de DEA Communication et Coopération dans les systèmes à agents, SMA/ENS Mines Saint-Etienne.

[BON00] Bonaventure, O. (2000) *Téléinformatique et réseaux: fonctions et concepts.* Facultés universitaires Notre-Dame de la paix de Namur.

[BRO86] Brooks, R.A. (1986) *A robust layered control system for a mobile robot.* IEEE Journal of Robotics and Automation.

[CHA01] Chaib-draa, B., Jarras, I. and Moulin, B. (2001) *Systèmes multi-agents: principes généraux et applications. Principes et architecture des Systèmes multi-agents.* Hermes.

[CHA02] Chaib-draab, B. (2001-2002) *Agents et systèmes multiagents.* Cours IFT-64881A, Laboratoire de recherche DAMAS, Dep. d'Informatique et de génie logiciel, Université LAVA, http://www.damas.ift.ulaval.ca/~coursMAS.

[DUR95] Durfee, E.H., Lesser, V.R. and Corkill, D.D. (1995) *Trends in Cooperative Distributed Problem Solving.* IEEE Transactions on Knowledge and Data Engineering.

[FER99] Ferber, J. (1999) *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence.* Pearson Education Limited.

[FEU71] Feuvrier, C.V. (1971) *La simulation des systèmes: maîtrise d'informatique.* Dunod Paris.

[FIC00] Fichefet, J. (2000) *Théorie des graphes et compléments de mathématiques.* Facultés universitaires Notre-Dame de la paix de Namur.

[HAD93] Haddadi, A. and Sundermeyer, K. (1993) *Knowledge about other agents in heterogeneous dynamic domains, International conference on intelligent and cooperative information systems, p.64-70.* IEEE computer society press Washington (D.C.).

[JEN98] Jennings, N.R., Sycara, K., Wooldridge M. (1998) *A Roadmap of Agent Research and Development.* Journal of Autonomous Agents and Multi-Agent Systems.

[JIA] Jiang, B., *Agent-based approach to modelling environmental and urban systems within GIS.* Division of Geomatics Institutionen för Teknik, Univerty of Gävle, Sweden.

[KIT99] Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A. and Shimada, S. (1999) *RoboCupRescue: Search and Rescue for Large Scale Disasters as Domain for Multi-Agent Research.* IEEE Conference on Man, Systems, and Cybernetics(SMC-99).

[MAG96] Magnin, L. (1996) *La simulation de RMC sur ordinateur.* Université de Paris.

[PAQ01] Paquet, S. (2001) *Coordination de plans d'agants: Application à la gestion des ressources d'une frégate.* Université de Laval.

[POU02] Poudade and Elkouby (Directeur de Recherche CNRS: Jean-Paul Sansonnet) (2002) *Langages de communication et d'échange d'informations entre agents.* Université de Paris, http://www.limsi.fr/Individu/jps/sma/poudadeelkouby/index.htm.

[PYN01] Pynadath, D. and Tambe, M. (2001) *An automated Teamwork Infrastructure for Heterogeneous Software Agents and Humans.* Information Sciences Institute and Computer Science Department, University of Southern California.

[QUI00] Quinqueton, J. (2000) *Systèmes multi-agents: une introduction.* Cours de l'Inria et Lirmm Montpellier.

[SPR82] Spriet, J.A. and Vansteenkiste, G.C. (1982) *Computer-aided modelling and simulation.* Academic press London.

[SUG99] Sugasaka, T., Masuoka, R., Sato, A., Kitajima, H. and Maruyama, F. (1999) *SAGE and Its Application to Electronic Commerce - SAGE:Francis: A System Based on "Virtual Catalog".* Systems and Computers in Japan.

[RAO95] Rao, A.S. and Georgeff, M.P. (1995) *BDI-Agents: from theory to practice.* Proceedings of the First Intl. Conference on Multiagent Systems (San Francisco).

[ROB00] RoboCupRescue technical committee (2000) *RoboCupRescue Simulator Manual version 0 revision 4.*

[WEB] http://cormas.cirad.fr/fr/demarch/sma.htm, Centre de coopération internationale en recherche agronomique pour le développement (CIRAD).

[WOO95] Wooldrige, M. and Jennings, N.R. (1995) *Intelligent Agents: Theory and Practice.* Knowledge Engineering Review.

[WOO98] Woolridge, M. and Jennings, N. (1998) *Agent Technology, Foundations, Applications, and Markets.* Springer, 1998, New York.

[YAB01] Morito, T., Kono, K. and Takeuchi, I. (2001) *YabAI The first Rescue Simulation League Champion.* Department of Computer Science, University of Electro-Communications, Tokyo, http://ne.cs.uec.ac.jp/~morimoto/.

VIII. ANNEXES

All our annexes are written on the CD you will find at the end of this document. You can also download all these annexes from our Web site: http://www.info.fundp.ac.be/~eleconte. Here follows the description of the different CD's component:

- Codes (java)

In this directory, you will find our source codes. There are two packages and they are already compiled. This will be very interesting for new agent developers to understand how the Simulation goes. These codes are runable with the *kernel* Ver. 0.31.

- Code documentation (html)

Here, you will find all the information (generated with javadoc) needed to clearly understand our codes.

- Manual V0r4 (in two formats: ps and pdf)

This is the official RoboCupRescue manual. Our thesis is filling certain lack of information of this guidebook, but it is still useful to attentively read it.

- Presentation (in three formats: ppt, ps and pdf)

Here is the presentation of our four mouths of work in Kobe. We present here all the steps of our work for the RoboCupRescue project. This could be useful too for new developers.

- Simulation movie (avi)

It is a recording of what the 2D *viewer* shows during the 300 turns of the Simulation.

- Thesis (in three formats: doc, ps and pdf)

Finally, you will find in this directory the electronic version of our thesis.