Annals of Software Engineering 13, 265–283, 2002
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

JAAFAAR: A Web-Based Multi-Agent Toolkit for Collective Research

S. CALDERONI and J.-C. SOULIÉ soulie@univ-reunion.fr
IREMIA – Université de la Réunion, 15, Avenue René Cassin BP 7151, 97715 Saint-Denis messag cedex 9, France

Abstract. This paper introduces a generic and opened multi-agent platform that provides a powerful scientific equipment for collective research on self-organized systems. A general thought on the mutation of object model towards the agent model is presented. The paper details the construction of the platform upon generic models of environment and agent. Then we present the extension of the toolkit to web-based technologies, and its suitability for collective research and remote works.

1. Introduction

This paper introduces a generic and opened multi-agent platform that provides a powerful scientific equipment for collective research on self-organized systems.

Computer simulations have became an undeniable tool for experimental research. On the one hand, many scientific domains like ecology, ethology, robotics or artificial intelligence needs simulation. On the other hand, software development is a complicated technical task.

We thus propose a multi-agent software platform to help scientists in designing artificial systems by modeling and simulating distributed artificial worlds with agent technology. JAAFAAR is a component-based system designed to build agent-oriented simulators from basic bricks as software units. It provides a generic model of environment that implements basic mechanisms inherent to multi-agent systems like signal propagation, phenomenology, agent perception and action.

The generic part of the system may be considered as an opened kernel provided with external interfaces. Each of these interfaces constitutes an entry point towards some abstract representation of the simulated world. Then, the design of specific domains consists in plugging some dedicated software components into these interfaces. Whether it is a simple object, an agent or one of its components like a sensor or an effector, or even an environmental phenomenon, such an entity is considered as a simple software component and may be added to the system with the same facility.

By another way, JAAFAAR is a web-based toolkit, founded on a client/server architecture, and provides an actual distributed experimental laboratory for remote scientists. Indeed, JAAFAAR permits to run a simulation on a server and offers entry points for remote clients that may concurrently act on the simulated world, in real time, throughout the Internet. Furthermore, it is possible to use the power of some remote computers by

distributing the reasoning process of agents that however evolve at the same time on the same environment. This allow scientists to confront heterogeneous agent architectures, without divulging their private components, throughout a common system. JAAFAAR is thus fully oriented for collective work.

The paper first presents a thought on the concepts that led the researchers to orient themselves towards agent technology, and how the object paradigm has brought a solid foundation to the implementation of multi-agent systems. Then, the heart of the platform is fully described, exhibiting both the environmental and the agent model of JAAFAAR, fundamental components of any multi-agent system. A case-base study is presented to illustrate the wheels of the platform. Finally, the web-side of the platform is presented. We examine the existing network technologies and explain how we enhance the functionalities of the platform towards web-based potentialities.

The kernel of JAAFAAR has been entirely implemented in Java language and some implementation details will be described in the paper.

2. From objects to agents

The increasing complexity of industrial softwares is due to the necessity to develop numerous sub-systems of various nature, including numerous functionalities and interacting with several human specialists (operators, experts, technicians, etc.) which are often spatially distributed. Consequently, it has became a necessity to split these systems in weakly coupled modules, namely in independent units where interactions are limited and totally controlled.

Thus, rather than being confronted to systems of monolithic architecture and structure, object oriented technologies have permitted to develop interacting software components which are locally defined and which don't have any global view of the system. Indeed, object-oriented programming proposes a different view of computer programs: a program becomes a set of entities that interact and communicate through messages sending. This conception brings up the autonomy of these entities called *objects*. The control of globality then becomes a secondary effect of local interactions. This form of program writing has introduced some new conception methods in software engineering and a perspective change; we have passed from notion of program to those of organization.

This evolution has also been observed in artificial intelligence, where intelligent systems were based on centralized architectures. It has became necessary to adapt machines and systems to the increasing complexity of problems to be solved. The manner to reduce this complexity was resting in the distribution of processes and knowledges. Intelligence was furthermore considered as the result of a cooperative activity between several simple entities, rather than the activity of a single omniscient and centralized system. The works led in the beginning of 70's on concurrency and distribution gave birth to a new research domain: Distributed Artificial Intelligence.

Distributed artificial intelligence then intends to remedy the insufficiency of the centralized approach of problem solving. The distributed approach proposes a distribu-

tion of processes on a set of systems able to interact in cooperation within a same environment (be they logical or physical), and to solve some possible conflicts. Research in distributed artificial intelligence is nowadays organized around three fundamental axes:

- Parallel Artificial Intelligence [Davis 1980], that focuses on development of efficient parallel languages and algorithms for distributed artificial intelligence.
- Distributed Problem Solving [Durfee *et al.* 1989], that focuses on the manner to distribute a given problem between cooperating entities.
- Multi-Agent Systems [Ferber 1994], that consist in putting in cooperation a set of autonomous entities called *agents*, which are provided with intelligent behavior, and in coordinating their goals and their action plans in order to solve a problem.

The approach that consists in studying, designing and implementing multi-agent systems was named *kenetic* by Ferber [1999]. The kenetic is both considered as the technique and the science of artificial organizations. It proceeds by constructing multi-agent systems, namely by implementing electronic or computational models composed of artificial entities that communicate and act in a same environment.

The term of *agent* is a pluridisciplinary term stem from specific vocabularies of interaction sciences like sociology, psychology, biology or ethology, and thus takes a signification of multiple facets. However, there exists some resemblances at the crossroads of these definitions, formulated by researchers from various domains. These resemblances all converge towards general concepts that could be drawn from the specificity of each science. Some researchers like Ferber applied themselves to formulate a minimal and common definition covering up the whole of these resemblances.

Definition 1. An *agent* is a physical or virtual entity that:

1. Is able to act in an environment.
2. Is able to directly communicate with other agents.
3. Is prompted by a set of tendencies (as individual purpose or a satisfaction function).
4. Holds some own resources.
5. Is able to locally perceive its environment.
6. Holds a local representation of its environment.
7. Holds competencies and offers services.
8. May eventually reproduce itself.

Distributed artificial intelligence, unlike classical artificial intelligence, takes an interest in the *interaction* that links agents among themselves, and agents with their environment. Agents do not merely reason, they *act*. Action is a fundamental concept for multi-agent systems, and relies on the fact that agents achieve actions that will alter their environment and thus their intended decision making. Communication is considered as

an extended form of action. The concept of intelligence is thus totally reviewed: rather than considering that it is only a property of cognitive faculties of an individual, we consider that it *emerges* from interactions that agents keep up among themselves and with their environment.

Agents hold not only the capability to act, but also the property of autonomy. Autonomy is here considered as the property of self-control. The agent is thus free to control itself by its own tendencies, that may take form of individual goals to satisfy or a satisfaction or survival function that it tries to optimize. This is certainly the fundamental difference between objects and agents.

Definition 2. A multi-agent system is a system composed of the following elements:

1. An environment E, namely a space generally provided with a metric.
2. A set of objects O. These objects are situated, this means that at any time, each object may be associated with a position in E. These objects are passive, this means that they may be perceived, created, destroyed and altered by the agents.
3. A set of agents A, which are particular objects ($A \subseteq O$). Agents are the active entities of the system.
4. A set of relationships R that link objects (and so agents) among themselves.
5. A set of operations Op that allow agents to perceive, produce, consume, transform and manipulate objects of O.
6. Some operators in charge of representing the application of these operations and the reactions of the world to this attempt of alteration, that we'll call the law of universe.

Among the numerous variable of multi-agent systems that meet this definition, two of them are typical:

- The fully *communicating* multi-agent systems, where $A = O$, $E = \emptyset$, and the relationships of R define a network: each agent is directly linked to a set of other agents, which one call its *acquaintances*. These systems are distinguishable from the others by the fact that interactions are essentially intentional communications and the working method is very similar to those of a social organization (working group, company, administration, etc.). The most famous example of such a system is certainly ARCHON,[1] that proposes a general architecture of multi-agent systems in order to integrate different applications which need to cooperate [Jennings and Wittig 1992].
- The fully *situated* multi-agent systems, where E is generally a metric and temporal space, perceptible by the agents, and that stands for their communication medium as a signal propagator. Agents, in this kind of systems, do not communicate directly by message sending, but only by signals propagation. The MANTA system of Drogoul, that turns on the modeling of the sociogenesis of an ant colony, is a splendid example of such systems [Drogoul *et al.* 1993].

[1] Architecture for Cooperating Heterogeneous On-line Systems.

Agent-based software engineering is a very large field of work and study at present. This is why there is a large number of platforms implemented in order to model multi-agent problems and run multi-agent simulations. In this set of platforms, we can cite the following ones:

- SWARM created by C. Langton [Minar *et al.* 1996] at the Santa Fe Institute (USA);
- CORMAS created by F. Bousquet and C. Lepage [Bousquet *et al.* 1998] at the CIRAD (France);
- MADKIT created by J. Ferber and O. Gutknecht [Gutknecht and Ferber 1997] at the LIRMM (France);
- the platform created by H.R. Gimblett and R.M. Itami [Gimblett and Itami 1997].

These platforms are used in various fields of research. For instance, CORMAS is largely used in Europe to simulate natural phenomena and the managment of natural ressources; SWARM has been used to implement ants algorithms from E. Bonabeau, M. Dorigo and G. Theraulaz [Bonabeau *et al.* 1999].

Whether it is in software engineering or in artificial intelligence, we can notice that the paradigm that gave birth to both objects and agents is fundamentally the concept of organization. In both cases, the necessity to tackle problems or programs of increasing complexity led the researchers to reconsider their systems from an organizational aspect. But from a practical point of view, the link that joins these two artifacts is furthermore intimate. Indeed, object model is undeniably an appropriate basis for the implementation of multi-agent systems. However, the mutation from object model to agent model has not been achieved directly. Actually, there is an intermediary model that links the two others: the *actor* model. One can define the actors as active objects, which are autonomous and concurrent. Actor models are particular object models able to dread the parallelism of actual systems. Actors are open to be implemented on parallel architectures. It is advisable to dissociate actors (active and autonomous objects) from the computational model of actors presented by Agha [Agha 1986]. Indeed, actors are stem from techniques of knowledge representation [Hewitt *et al.* 1973], then from parallelism since the publication of Act1 language in 1981 [Liberman 1987]. The main common features are the use of *scripts* (to describe the behaviors of actors) and *acquaintances* (to represent the attributes). The terminology of actor models draws nearer to living creatures than those of the initial object model. Furthermore, it is more easy to conceptualize the parallel interactions of several actors.

After the conceptual introduction of agent paradigm presented in this section, we now present in the next section how we have applied these concepts and implemented a powerful multi-agent platform with object-oriented technology.

3. JAAFAAR: a multi-agent platform...

3.1. What is JAAFAAR*?*

JAAFAAR is a generic multi-agent software toolkit designed with the aim to help re-

searchers in modeling and simulation of artificial worlds with agent technology. The thought of developing such a platform came to our mind by discovering the double clearness that – on the one hand, many scientific domains like ecology, ethology, robotics or artificial intelligence need the use of computer simulation, since it has became an undeniable tool for experimental research – and on the other hand, software development is a complicated technical task. Indeed, computer simulation provide a powerful device for virtual experiments, as opposed to physical ones which are difficult to design, costly, or even worse dangerous. However, although computer models provide many advantages over traditional experimental methods, some problems may be encountered too. In particular, the actual process of writing software is a complicated technical task with a high risk of error. This fact has brought us to design a powerful platform in order to allow scientists to focus on their research rather than on building appropriate tools. This platform is intended to provide a complete software environment viewed as a virtual laboratory and providing agent-oriented support to model and simulate autonomous artificial systems.

Another established fact induced us to extend the potentialities of JAAFAAR. At the time of Internet and computer networks, we no longer can think *globally*. Indeed, the computational space has became a huge *world wide web* in which any computer, be it stationary, portable or mobile, is connected to all over the world. Data and processing power are distributed on a considerable number of sites. Why not use this potential power? It may be very advantageous to distribute an heavy application among several computers interacting among themselves through Internet. Furthermore, this new dimension opens the way to collective research and brings the possibility of remote work. JAAFAAR is thus more than a multi-agent platform. It is also a distributed experimental laboratory for remote scientists. Finely speaking, JAAFAAR is based on a client/server architecture, where the simulated world run on a server that can be reached by clients like monitors for the remote control of the system, or remote agents.

We will now detail the software conception process we have adopted to design JAAFAAR at a generic level, and explain how it is possible to use this generic platform to design a dedicated multi-agent application. With this purpose, we chose to describe a typical multi-agent problem in order to illustrate the different wheels of JAAFAAR. This problem stages a colony of autonomous robots in a collective regulation problem and is detailed on the next section.

3.2. Mining colony on Tatooine: a collective regulation problem

A few month ago in a galaxy far, far away . . . a group of settlers were sent to Tatooine, a world of brilliant crystalline sands that revealed itself being a treasure trove of minerals and ores. Mining colonies have been established in a relatively small area of the planet's Northern hemisphere. An unusual concentration of magnetic ores in the planet's mantle, interacting with Tatooine's intense planetary magnetic fields, shifts prevailing wind patterns and atmospheric concentrations, creating a zone of relative coolness in one location. While the remainder of the planet reaches highs better than 65.5°C throughout

the year, the single temperate zone rarely exceeds 43°C, while reaching nighttime lows near freezing. Although little exploration has mapped out all of Tatooine's surface, there are no outstanding features or mining opportunities enticing enough to cause habitation of the planet's hotter regions.

Since the air is almost unbreathable, because of the presence of some poison gas, the settlers were forced to build a biosphere providing an artificial atmosphere. They tuned a novel chemical process that can produce oxygen from the transmutation of large amounts of quartz (75%) and sodium salts (25%), both present in the sand-rich soil of Tatooine. They built four types of autonomous robots, explorers, borers, carriers and tenders, responsible for the prospecting task:

- The *X-PLORER*S are in charge of exploring task. These robots are provided with ore veins sensors. Their task consists in wandering through the environment until they find a vein. When a vein is localized, they have to emit a recruiting signal to warn the borers and the carriers. Furthermore, they are in charge of supervising the mining task to detect the exhausting of ore.
- The *D-GER*S are in charge of mining task. These robots have to join the explorers when they detect their recruiting signal. Then they have to bore the soil to extract ore samples while they perceive the recruiting signal (which indicates that the vein is not exhausted). They are provided with a chain arm to pick up ore samples. When carriers are close by, they put down the extracted samples in their tub. Their task stops when the local explorer interrupt its signal because of the exhausting of ore.
- The *S-CARGO*S are in charge of conveying task. These robots have to join the explorers when they detect their recruiting signal, like the borers. They are provided with a big tub to stock the extracted ore samples. They wait the borers fill in their tub, then they convey their loading to the base.
- The *10-DER*S are in charge of supplying task. Actually, each robot needs and consumes energy while it performs an action. This energy derives from an electrochemical process between the gas present in the Tatooine's atmosphere and the oxygen produced in the biosphere. Thus, the robots are all provided with a tank of liquid oxygen with capacity depends on their respective task. When a robot lacks of oxygen, it emits distress signal to wander tenders it needs gas supplies. Thus, the supplying task of each tender consists in joining any robot emitting a distress signal and supplying it with a sufficient amount of liquid oxygen (if available). When its own tank is empty, it has to return to the base to fill up.

The figure 1 illustrates the activity of the robots in the mining colony of Tatooine.

At the global scale, it is essential that the total amount of quartz and sodium salts held in the reserve tank of the base never decrease under a minimum threshold to preserve the viability of the biosphere. Additionally, it seems obvious to minimize the amount of oxygen consumed by the robots. Thus, the robots are confronted to a dilemma: they have to keep a sufficient amount of ore in the base's reserve tank required to produce the vital oxygen of the biosphere, and in the same time, they need to consume a part of this oxygen to achieve their tasks. So, they are faced with a regulation problem.

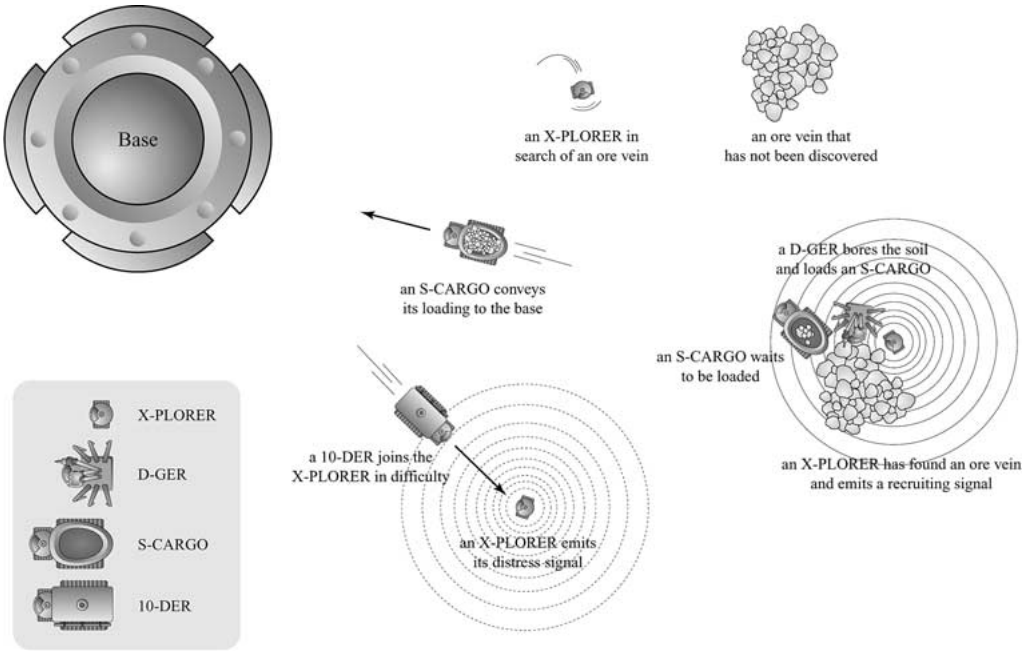

Figure 1. The mining colony on Tatooine.

Due to the specificity of each task and structural constraints, it was impossible to design omniscient robots, able to achieve each step of the global prospecting task. Each robot is then specialized in a specific sub-task, and has been provided with all individual behaviors required for its local purposes.

However, these individual capabilities are insufficient to ensure the global regulation task. Indeed, the robots cannot play their respective roles without coordination mechanism. They are all capable of ensuring sub-goals, but most of them depend on the others to achieve their local purposes. For instance, a D-GER is unable to perceive the presence of ore, and needs the help of an X-PLORER to show it the good location to bore. Also, it would be tedious and highly energy-consuming for a single D-GER to convey the samples it has extracted with its sole chain arm. Thus, it needs the assistance of an S-CARGO which is provided with a tub of high capacity. Furthermore, each robot is depending on 10-DERS which are the sole able to supply it with liquid oxygen when it works outside.

Actually, it is vital that robots respect a coordinating policy in their behavior to secure the whole group, but additionally, they must keep their own survival, which is also important for the group subsistence. Indeed, an X-PLORER must emit a recruiting signal when it has discovered an ore vein to warn the others, but the action of emitting is costly since it requires energy. Now, if no D-GER arrives on site, the explorer risks to expense all of its reserves, preventing it to get back the base, therefore it must be adaptive and able to detect the problem before it is too late! Indeed, a robot whom has emptied its tank is unable to perform anything and stops immediately. The only thing

that may help it is then the spontaneous assistance of a 10-DER.

In summary, each robot must keep its integrity by adopting a survival behavior while it must act in a social way through a collective commitment to ensure the coordination of the group at each local step required for the global regulation constraint. In other words, it has to balance its responses in reaction to local influences that condition its behavior, in such a way that the whole of robots maximize its performance regarding to the global task.

Now that the background has been explained, we can launch out into the conception of JAAFAAR. The architecture of the platform is directly inspired of definitions 1 and 2 laid down in section 2 and is organized around a generic environmental model, and a generic agent model.

3.3. The environmental model

The environment forms as it were the heart of the multi-agent system, since it is the exclusive medium of interaction. Consequently, it is the essential link between agents. The term of *interaction* indicates the mutual influence that an individual has on another one. This concept of inter-individual interaction denotes the basis phenomenon that we should use to describe the functioning of a social group. Any social mechanism should thus amounts to a set of elementary interactions. Then, the environment is the support of the interaction phenomenon. The first thing to consider is how to give a substrate to interactions. Influences between two agents are exerted through the combination of perception and action. The exchange of information between perception and action processes is materialized by the emission and propagation of signals in the environment.

Let us take the example of an agent A that perceives the motion of an object B. The motion of B can be disclosed by A by the recognition of its change of location in the environment. This analyze of such a change is only possible if objects leave a sort of trail in the environment. In other words, each object should emit a recognition signal to be perceived. Then the analyze of the variation of such a signal can reveal the result of an action from the observer point of view. Suppose that the agent A is a robot, equipped with vision sensors, and the object B is another robot that moves within the perception field of A. The sensors of A, which are sensitive to light waves reflected by B (this is the signal), can inform A on the nature of the B's motion. Actually, the emitted signals holds some properties of which some will be identified and valued by sensors. Practically, signals propagation is entrusted to specific propagators (all instances of a `Propagator` class) depending on their respective nature which directly affect the sensors.

Let us consider our autonomous robots on Tatooine, and especially the recruiting process by which X-PLORERS attract D-GERS by emitting a "recruiting" signal. The environment is provided with a propagator dedicated to the propagation of signals whose signature is of "recruiting" type, namely an instance of the `jaafaar.environment.Propagator` class. Each propagator is specific to a single type of signal because the propagation process may differ according to the signal's nature. For example, some signals may be able to go through obstacles (like radio

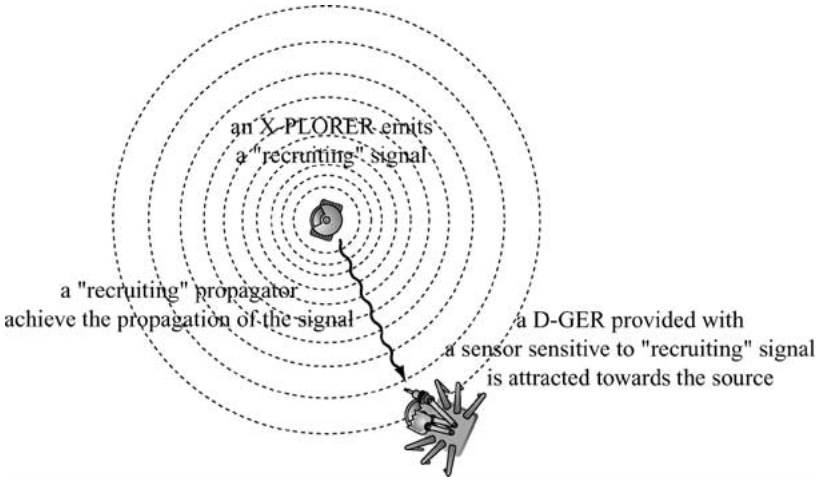

Figure 2. Interaction between agents through signal propagation.

waves), whereas others are stopped (like light waves). Consequently, the environment has to be provided with as much propagators as signal types.

The function of any propagator consists in scanning the set of agents that hold the good sensor to perceive the signal that is to be propagated. Then, for each selected agent, the propagator has to transmit to its sensor the properties of the signal which are perceptible (e.g., intensity, direction, distance, etc.). One of the most important thing to examine here is how to format this information, since this is precisely *on this information* the agent will found its local perception of the world, and then its behavior. This is precisely this information that makes the link between the state of the environment and the agent. We chose to represent this information as a lisp-like symbolic description of the following form:

(signal_type emittor_id (property$_1$ value$_1$) ...
(property$_n$ value$_n$)).

Obviously, another way to represent is the use of ACLs languages such as KQML [Finin *et al.* 1993]. But our aim, when developping JAAFAAR, is to produce an esay to use platform for researchers and scientists. Indeed, this is not very easy to learn and use ACLs languages for researchers and scientists coming from other research fields such as biologists, genetitian, ... Our lisp-like language is easy to use and its symbolic is very expressive, so that it can be easily learn. This description is given as a `java.lang.String` to facilitate its transmission over the network in the case of distributed applications. Such symbolic expressions are then easy to parse, due to their lisp-like syntax.

In the previous case of recruiting process, if we consider that the environment is provided with a 2D-metric, the perceived information could be of the following form (where distance is relative to the sensor and expressed in meters, and direction is relative to the agent orientation and expressed in radians):

```
(recruiting explorer_4 (distance 20.68) (direction -0.74)).
```

We can notice with this example, that *communication* is a specific interaction process that can be implemented as a signal propagation.

Generally speaking, we could draw together the notion of signal to those of *field*. In physics, be they gravitational, electric or magnetic, fields can explain the origin of powers and their influence on particles. In the same way, signals can be considered as stimuli, and then stimuli fields can explain the influence of environment (or its components) on agents. We generally distinguish two types of field: scalar fields and vectorial fields. These fields associate to each point of the space a value. In the case of scalar fields, this value is a scalar, which may represent the intensity (the potential) of the field in this point, whereas, in the case of a vectorial field, this value represents a vector. These fields may be combined and superposed if they are of same type. The visualization of a scalar field's intensity in accordance with its location in a two-dimensional space may be illustrated by a surface with peaks in high intensity regions and with valley where the intensity is low. The slopes of these "mountains" correspond to the field's gradients, that is to say its variation. The gradient is a vectorial function that associates a vectorial field to the initial scalar field. Where the scalar field defines a potential field, the gradient field defines a power field that corresponds to the field's potential. These fields then determine some attractive and repulsive tendencies that condition the behavior of agents.

Outwards interaction processes, the environment (and, consequently, its components) may be altered by what we name a *phenomenology*. A phenomenon is considered, in JAAFAAR, as a process that may alter the properties (namely the state variables) of any object in the environment. For example, motion may be considered as a phenomenon since it is a process that modifies the kinematics properties of any object affectable by motion (namely its position, velocity and acceleration). Phenomena are described by derived classes of the abstract class `jaafaar.environment.Phenomenon`. The function of any phenomenon P consists in scanning the set of objects that are affectable by P, and applying some operations on each object's state. Let us consider the case of a 2D-environment where objects are movable. Thus the environment has to be provided with a `Motion2D` phenomenon that inherits from the `jaafaar.environment.Phenomenon` class. All objects are provided with the following time-dependent kinematics 2D-vectors: position $\vec{p}^t$, velocity $\vec{v}^t$ and acceleration $\vec{\gamma}^t$. The *decay* factor represents frictions embodied by the environment. It alters the objects velocity. This factor is simulation dependent: indeed, it can increase the velocity (i.e., $decay > 0$) or reduce the velocity (i.e., $decay < 0$). At each simulation step, movement of each object is calculated by the `Motion2D` phenomenon as the following manner:

$$
\begin{aligned}
&\vec{u}^{t+1} = \vec{v}^t + \vec{\gamma}^t, \\
&\vec{p}^{t+1} = \vec{p}^t + \vec{u}^t, \\
&\vec{v}^{t+1} = decay \cdot \vec{u}^{t+1} \quad \text{for a damped movement,} \\
&\vec{\gamma}^{t+1} = \vec{0} \qquad\qquad\quad \text{to reset acceleration.}
\end{aligned}
$$

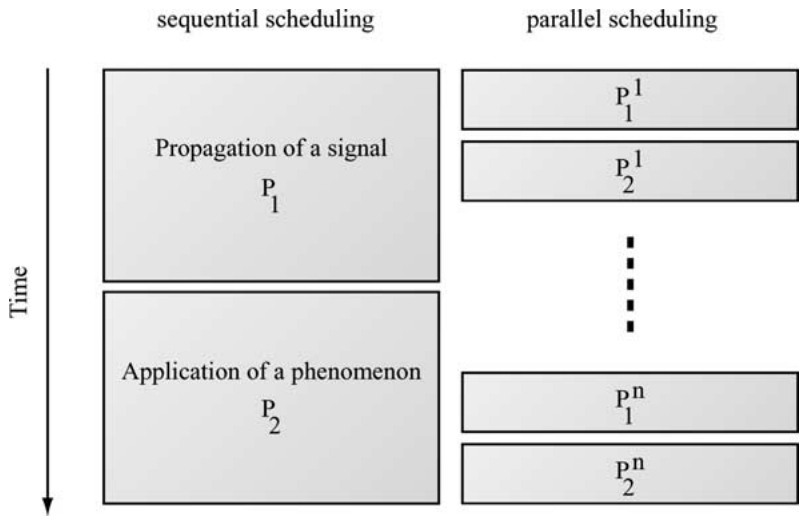

Figure 3. Parallelization of phenomena and signal propagation processes.

It is important to notice that phenomenology and signal propagation are time-continuous processes that must be discretized to be implemented. Indeed, it is not acceptable to wait for the end of such a process to apply the effect of another one. All processes must be concurrent in time, and thus performed in a parallel manner. This imply that processes effect must be discretized (this means sliced) in time, and executed concurrently as shown on figure 3.

We achieved this functionality in JAAFAAR by providing the environment with a *clock* that gives the rhythm to the simulation. This clock must be an independent mechanism, asynchronous and concurrent with the rest of the system, thus it has been derived from the `java.lang.Thread` class. At each time step, a mechanism activates a scheduler that successively performs the application of all discretized phenomena and the propagation of all signals throughout the environment.

3.4. The agent model

As we had developed it in the environmental model, interindividual interactions are only possible if agents are equipped with sensors to perceive any environmental information, and effectors to act on the environment. Furthermore, the coordination between perception and action should be achieved by a control system, that stands for the *brain* of the agent, and permits it to choose which action to accomplish, at what time, in accordance with its goals. Thus, we can split the notion of agent into two complementary parts:

- the brain, that is entirely independent from the environment and constitutes its *soul* as it were,
- and the *body*, that corresponds to its physical part, namely those that exists within the environment.

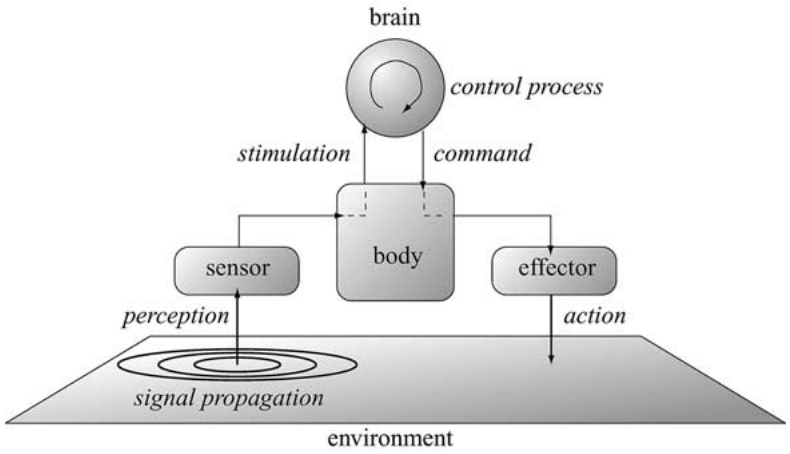

Figure 4. The JAAFAAR agent architecture.

The body of the agent constitutes the entity that links it to the environment. Indeed, this is precisely its body that is provided with sensors for perception and with effectors for actions. On the one hand, the signals perceived at the sensors level are redirected by the body towards the brain as *stimulations*. In a symmetrical manner, the brain addresses *commands* to the body which are communicated to effectors to be transformed into actions. The figure 4 illustrates the information flow exchanged between the agent and the environment.

Concretely, at each simulation step, the signals propagators write information (as symbolic descriptions), corresponding to perception, in body's sensors. These descriptions then forward by the body towards the brain as stimulations. In the same time-step, all the commands addressed by the brain are stored in the corresponding effectors by the body. In this way, some environmental processes can read these commands and transform them into effective actions. Let us take the example of a robot that wishes to move forward. It may formulate a `boost` command with appropriate parameters (e.g., the associated power). This command is expressed as a symbolic expression for the same reason as the perceptions: `(boost 100)`.

By another way, the body also encloses the state variables of the agent, exactly as the other objects. These state variables constitute a representation of the agent within the environment. This is precisely these variables which are susceptible to be altered by environmental phenomena. We saw, in the previous section, how an object (and thus an agent) can be affected by a motion phenomenon through its kinematics properties. In the case of the transformation of a command into an action, the process is exactly similar. In the previous example of a robot that wishes to move forward, the command `(boost 100)` is transformed into an action by setting up (in this case) the acceleration vector of the robot with a function of the power parameter of the `boost` command.

Now we have detailed the interactions between the agent and the environment, let us focus on the autonomous part of the agent, namely the part that shapes its behavior: its brain.

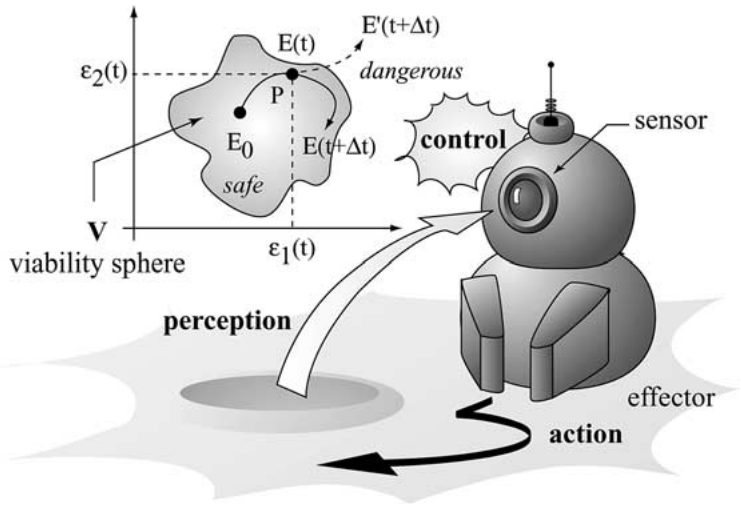

Figure 5. An adaptive perception–control–action triad.

The behavior of an agent may be qualified of adaptive while its control system is able to keep its essential variables within their *viability sphere*. The figure 5 represents a robot moving on a ground scattered with holes. Its brain should help it to avoid these holes in order to preserve its integrity. Its state $E(t)$ may be expressed in terms of two variables $\varepsilon_1(t)$ and $\varepsilon_2(t)$ that vary over time. The viability sphere V may be considered as a region of the states space where E should be kept. The outer side of V corresponds to states that may endanger the survival or the goal of this robot. The sensors of the robot should indicate to itself that there is a hole in front of it, and its control system then has to choose which action to accomplish. Thus, at point P, $E(t)$ risks of leaving V in $E'(t + \Delta t)$ if the agent moves towards the hole. An adaptive behavior would be the one that performs a corrective action on this state transition so as to transform E in $E(t + \Delta t)$, which is inside of V, by passing round the hole.

A detailed presentation of architectures for such adaptive control systems is out of the scope of this paper. Nevertheless, the interested reader may refer to [Calderoni and Marcenac 1998], that presents MUTANT, an evolutionary-based learning system for adaptive control systems.

In conclusion to this section, what the reader should remember is that it is very easy to design agents with the JAAFAAR toolkit. Indeed, all the underlying mechanisms inherent to agent paradigm are already implemented, and the agent structure can be summarized to only two generic classes:

- `jaafaar.agent.Brain`. This class is an abstract class and stands for the basis to develop your own brain architecture. You are free to design you own learning algorithm, your own control system by inheriting this generic class.
- `jaafaar.agent.Body`. This class is also an abstract class, and stands for the basis to develop your own body architecture. This means you should inherit this

class to specify which kind of signal your agent is able to propagate, which kind of phenomenon is able to affect your agent, which kind of sensor and effector you want to provide your agent with.

4. ...For collective research

4.1. Why?

We have decided to extend the potentialities of JAAFAAR to the web-space by stating the following facts:

- First of all, the dynamic of international research implies a constant challenge among researchers. They need to confront their works constantly. The primary vectors of scientific publications are obviously journals and conference proceedings. However, with such sources of works reports, researchers are able to confront their ideas, but rarely the way they have concretized them. In other words, the technical aspects of any research is rarely described in such a way that they may be reproducible as they stand. Especially in computer science, the details of implementation often occulted because of the lack of room.
- A first alternative is to try to re-implement the described system, by interpreting its hazy descriptions. But the resulted system may present some differences with the original, and consequently the comparison with your own works risks to be falsified, because you are comparing *your* interpretation of the desired system. Furthermore the process of writing software is time-consuming and presents a high risk of error for those who are not familiar with solid software engineering methodologies.
- A second alternative consists in contacting the author and ask him to send you its own software or its source code so that you will be able to adapt it to your needs. However, many researchers refuse to release their applications, or furthermore to divulge their source codes. And in the case they do it, the software may be difficult to compare with yours as it stands or even not suited to your operating system, or the source code may not correspond to the language you usually use. Then you risk to find yourself in the previous position.
- An ideal alternative would be to directly connect the systems to be compared within a same structure, and make them interact. Actually, at the time of Internet, it seems inconceivable that one cannot use the power of networks to link remote systems to do so.

We thus decided to propose such a possibility in JAAFAAR by coupling our ability in generic platform development with the power of web-based technologies. The solution we propose is to make JAAFAAR a web-based toolkit, founded on a client/server architecture, in order to provide researchers with an actual distributed experimental laboratory for remote work. Our idea is to propose an opened system able to receive heterogeneous and physically distributed software components from the Internet to be confronted

among themselves. In other words, we have provided JAAFAAR with the capability to receive remote agents from the Internet. Furthermore, we have added the possibility to connect some remote monitors to allow remote researchers to control the evolution of a simulation without being physically present on the site that hosts the simulation.

4.2. How?

To achieve our purpose, we first have had to examine the existing technologies to add some network functionalities to JAAFAAR. The most famous of them are undeniably – RMI,[2] which is a remote control extension of Java for distributed applications – and CORBA,[3] which is a standard for remote control of common objects through the Internet. In the case of RMI, we risk to limit the platform to interact exclusively with Java applications, whereas in the case of CORBA, we open the possibility to interact with object-based applications that however must implements this norm. In both cases, we are limited to interact with object-based systems. This can be unacceptable since many researchers use non-object-oriented language to implement their prototypes. That's why we chose to found our network extension of JAAFAAR to more basic protocols whose are widely supported in the most common languages, be they object-oriented or not.

Despite of this established fact, we can nevertheless learn lesson from these fashionable technologies [Orfali and Harkey 1997]. Both RMI and CORBA offer functionalities to transparently control and invoke methods on objects, be they local ore remote to the system. This is achieved thanks to three basis components:

- The local and instantiable objects of the system, named *skeletons*.
- The remote and not instanciable objects which are the counter-parts of the skeletons, named *stubs*.
- The channel that links the two previous entities for communication, named ORB[4] in CORBA technology.

To transpose these concepts into JAAFAAR then consists in binding local and remote components by a communication channel with a communication protocol. More precisely, the remote components that may be connected to a JAAFAAR simulator are either monitor applications for remote control of the simulator, or remote agents which are actually remote brains. In both cases, these components are clients which can be linked to the server by socket[5] connections.

One of the primordial steps in the design of client/server applications consists in choosing the kind of socket used over IP (the level 3 in the OSI[6] Model that contains seven levels). The Java Development Kit from SunMicrosystems provides two kinds of built-in protocols:

[2] Remote Method Invocation.
[3] Common Object Request Broker Architecture.
[4] Object Request Broker.
[5] A socket is a network communication interface.
[6] Open Systems Interconnection.

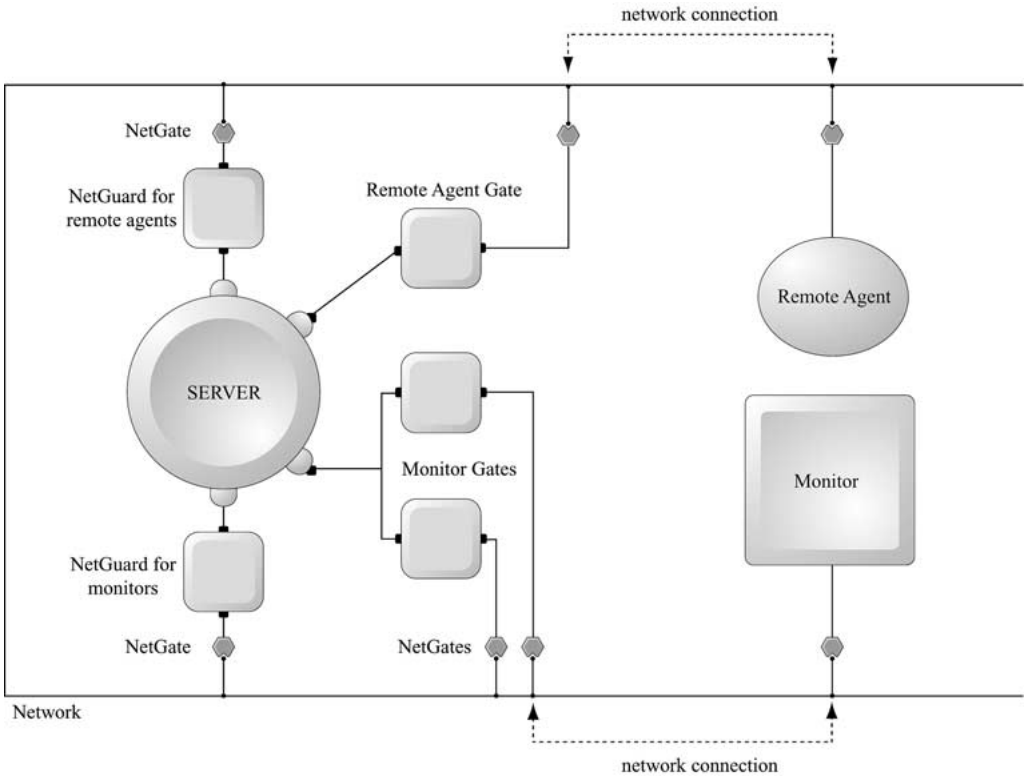

Figure 6. The network architecture of JAAFAAR.

- UDP,[7] which is a connectionless protocol that runs on top of IP networks. UDP/IP provides very few error recovery services, offering instead a direct way to send and receive datagrams over an IP network. It's used primarily for broadcasting messages over a network. UDP is useful when TCP would be too complex, too slow, or just unnecessary.
- TCP,[8] which is the suite of communications protocols used to connect hosts on the Internet. TCP/IP uses several protocols, the two main ones being TCP and IP. TCP/IP is built into the UNIX operating system and is used by the Internet, making it the de facto standard for transmitting data over networks. Even network operating systems that have their own protocols, such as Netware, also support TCP/IP. TCP is useful when you want to be sure that a packet send over the network is well arrived.

Due to the efficiency of UDP with respect to TCP and because an eventual loss of data is not prejudicial to our applications, we have chosen to use UDP sockets rather than TCP. Moreover, according to expected simulations, the speed of data transfer can be crucial, then the speed is the more essential parameter. To implement UDP sockets, Java

[7] User Datagram Protocol.
[8] Transfer Control Protocol.

provides two dedicated classes [Flanagan 1997]: `java.net.DatagramPacket` and `java.net.DatagramSocket`. Sockets represent the network communication interfaces, whereas packets represent the messages which are in transit on network between two sockets.

Although Java provides some low-level mechanisms to exchange messages on the network using UDP sockets, we have had nevertheless to implement the high-level connection between sockets and our distributed software components. This was achieved by implementing higher-level network communication interfaces which we named `jaafaar.net.NetGate`. Actually, a `NetGate` embeds a `DatagramSocket` and is able to transmit some `java.lang.String` over the network by transforming their encoding description into `DatagramPackets`. This gave us the freedom to develop a small command language as communication protocol between our software components: JaafaarTalk, whose syntax is based on lisp-like symbolic expressions.

Thanks to these `NetGates`, any JAAFAAR server is then able to communicate with any distributed applications (remote clients) using JaafaarTalk as communication protocol, and this whether they are written in `C`, `C++`, `Ada`, `SmallTalk`, `Java`, or in any language that support the UDP protocol. The figure 6 finally illustrates the network architecture of JAAFAAR.

5. Conclusion

This paper has presented the features of JAAFAAR, a web-based multi-agent toolkit for collective research. We first have explained how the enhancement of the object model led to the agent model. Then we have presented the heart of the platform as a multi-agent toolkit founded on both an environmental model, which constitutes the exclusive interaction medium between agents and supports signal propagation and phenomenology, and an agent model, that includes perception/action mechanisms and a behavior control capability. We have then point out the utility to extend the potentialities of JAAFAAR to web-based technologies. We have explained how we made our platform an actual distributed experimental laboratory for remote and collective research. Indeed JAAFAAR takes into account not only internal components written with Java, but also networked components written with any type of language, be they object-oriented or not. The remote agents have a physical representation within the JAAFAAR's environment (their body), and remote users can control their behavior with remote brains and observe the simulation with remote monitors. Moreover, since more than one remote agent can be connected to JAAFAAR, researchers can then confront their own architectures to others and, consequently, enhance the speed of their works and preventing them from errors due to bad implementations. Research in multi-agent systems needs such toolkit to test and validate agent architectures on commonly defined problems. JAAFAAR opens this possibility with simpleness. Finally, we plan to introduce this platform in our teaching as a challenging framework to emulate the creativity of our students through agent contests.

References

Agha, G. (1986), "Actors: A Model of Concurrent Computation," In *Distributed Systems*, Artificial Intelligence, MIT Press.

Bonabeau, E., M. Dorigo, and G. Theraulaz (1999), "Swarm Intelligence: From Natural to Artificial Systems," *Journal of Artificial Societies and Social Simulation 4*, 1.

Bousquet, F., I. Bakam, H. Proton, and C.L. Page (1998), "CORMAS: Common-Pool Ressources and Multi-Agents System," In *Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, A. del Pobil and M. Ali, Eds., Lecture Notes in Artificial Intelligence, Vol. 1416, Springer, Berlin, pp. 826–838.

Calderoni, S. and P. Marcenac (1998), "MUTANT: a Multi-Agent Toolkit for Artificial Life Simulation," In *Proceedings of the 26th International Conference on Technology of Object-Oriented Languages and Systems*, M. Singh, B. Meyer, J. Gil, and R. Mitchell, Eds., IEEE Computer Society Press, Santa Barbara, CA, pp. 218–229.

Davis, R. (1980), "Report on the Workshop on Distributed Artificial Intelligence," *SIGART Newsletter 73*, 42–43.

Drogoul, A., B. Corbara, and D. Fresneau (1993), "Manta: New Experimental Results on the Emergence of (Artificial) Ant Societies," In *Simulating Societies Symposium*, C. Castelfranchi, Ed.

Durfee, E.H., V.R. Lesser, and D.D. Corkill (1989), "A Survey of Cooperative Distributed Problem Solving," In *The Handbook of Artificial Intelligence*, A.B. Barr, P.R. Cohen, and E.A. Feigenbaum, Eds., Vol. 4, Chapter 17, Addison-Wesley, Reading, MA, pp. 83–147.

Ferber, J. (1994), "Reactive Distributed Artificial Intelligence: Principles and Applications," In *Sixth Generation Computer Technology*, G. O'Hare and N. Jennings, Eds., Wiley-Interscience Publication, New York, pp. 287–314.

Ferber, J. (1999), *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, Addison-Wesley, Reading, MA.

Finin, T., J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, R. Pelavin, S. Shapiro, and C. Beck (1993), "Specification of the KQML Agent-Communication Language," Technical Report, The DARPA Knowledge Sharing Initiative – External Interfaces Working Group.

Flanagan, D. (1997), *Java in a Nutshell (Version 1.1)*, Second Edition, O'Reilly.

Gimblett, H.R. and R.M. Itami (1997), "Modelling the Spatial Dynamics and Social Interaction of Human Recreators Using GIS and Intelligent Agent," In *Proceedings of the International Congress on Modelling and Simulation*, Hobart, Tasmania.

Gutknecht, O. and J. Ferber (1997), "MadKit: Organizing Heterogeneity in a Platform for Multiple Multi-Agents Systems," Technical Report LIRMM–97189, Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier.

Hewitt, C., P. Bishop, and R. Steiger (1973), "A Universal Modular ACTOR Formalism for Artificial Intelligence," In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford, CA, pp. 235–245.

Jennings, N.R. and T. Wittig (1992), "ARCHON: Theory and Practice," In *Distributed Artificial Intelligence: Theory and Practice*, ECSC, EEC, EAEC, pp. 179–195.

Liberman, M. (1987), "Concurrent Object-Oriented Programming in Act1," In *Object-Oriented Concurrent Programming*, Yonezawa, Ed., MIT Press, Cambridge, MA, pp. 9–36.

Minar, N., R. Burkhart, C. Langton, and M. Askenazi (1996), "The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations," Technical Report, Santa Fe Institute.

Orfali, R. and D. Harkey (1997), *Client/Server Programming with Java and Corba*, Wiley Computer Publishing, New York.