Institut de REcherche en Mathématiques et Informatique Appliquées

LISTE DE TRAVAUX

2

Université de la Réunion

15, avenue René Cassin - BP 7151 - 97 715 Saint-Denis messag cedex 9
Téléphone : 02 62 93 82 82 Télécopie : 02 62 93 82 60
E-mail : iremiase@univ-reunion.fr
http : //www.univ-reunion.fr

2 000

2

UNIVERSITÉ DE LA RÉUNION

LISTE DE TRAVAUX

2

Pierre Marcenac

HABILITATION À DIRIGER DES RECHERCHES

Réf≠ : PM/HV4W/9709
N° d'ordre :

Les travaux présentés dans ce dossier sont les suivants :

[Marcenac et Giroux 98] : P. Marcenac, S. Giroux, "GEAMAS: A Generic Architecture for Agent-Oriented Simulations of Complex Processes", International Journal of Applied Intelligence, à paraître en 1998.

Cet article décrit en détail le coeur du travail, le modèle et l'organisation d'agents proposés, ainsi que l'une des expérimentations permettant d'en valider les principes (application à la simulation d'éruptions volcaniques).

[Marcenac 98b] : P. Marcenac, "Modeling MultiAgent Systems as Self-Organized Critical Systems", 31st Hawaii International Conference on System Sciences, HICSS-31, IEEE Computer Society Press, à paraître en Janvier 1998.

Cet article focalise sur l'étude des mécanismes intrinsèques des agents, en particulier l'auto-organisation constituant un point clé du modèle proposé. Il présente en outre une seconde expérimentation montrant l'application de ces principes (application à la sismicité).

[Marcenac 98a] : P. Marcenac, "A computer-simulation tool for evolutionary systems", International journal of knowledge-based intelligent engineering systems, à paraître en 1998.

Cet article présente l'environnement de développement GEAMAS V2, issu de l'abstraction des applications et entièrement reconçu et redéveloppé an Java au cours de l'année 1997. Il vient compléter les deux articles précédents, en focalisant sur les aspects d'analyse, de conception et d'implémentation de l'environnement logiciel proposé.

[Leman et al. 96] : S. Leman, P. Marcenac, S. Giroux, "Un modèle multi-agents de l'apprenant", Revue AFCET STE, Sciences et Techniques Éducatives, Hermès, Vol. 3, N°4, 1996, Pages 465-483.

Enfin, cet article présente une vue globale de la première expérimentation menée dans le cadre de nos travaux, l'application aux environnements d'apprentissage avec ordinateur (thèse de S. Leman), qui a permis de valider la représentation en organisation d'agents, et une partie du modèle d'agent.

GEAMAS: A Generic Architecture for Agent-Oriented Simulations of Complex Processes

Pierre Marcenac, Sylvain Giroux

[Marcenac et Giroux 98]

APIN572-96 Accepted Paper

To be printed in International Journal of Applied Intelligence, 1998.

GEAMAS: A Generic Architecture for Agent-Oriented Simulations of Complex Processes

Pierre Marcenac[1], Sylvain Giroux[2]

Abstract:

This paper's object is to present the results of the GEAMAS[3] project which aims at modeling and simulating natural complex systems. GEAMAS is a generic architecture of agents used to study the emergence of the behavior of such systems. It is a multiagent program meant to develop simulation applications. Modeling complex systems requires to find a way to reduce, to organize the system complexity and to describe suitable components. Complexity of the system can then be tackled with an agent-oriented approach, where interactions lead to a global behavior. This approach helps in understanding how non-determinist behavior can emerge from interactions between agents, which is near of self-organized criticality used to explain natural phenomena. In the Applied Artificial Intelligence context, this paper presents an agent software architecture using a model of agent. This architecture is composed of three abstract levels over which the complexity is distributed and reduced. The architecture is implemented in ReActalk, an open agent-oriented development tool, which was developed on top of Smalltalk-80. To illustrate our purpose and to validate the architecture, a simulation program to help in predicting volcanic eruptions was investigated. This program was run over a period of one year and has given many satisfying results unattainable up to there with more classical approaches.

1. INTRODUCTION

The ability of natural phenomena to cause damage upon the earth each year has recently led to their modeling in computer systems and has become a privileged application of artificial intelligence. To understand such physical worlds, simulation applications have been developed to help in exploring the complexity of underlying phenomena on large and small scales. Simulation is interesting, because it can adequately capture any behavior likely to be observed, and is used to exhibit new parameters and to tackle their role in the appearance of natural phenomena.

In order to explore the emergence of such behaviors, this paper tackles the issue of modeling and simulating complex systems using an agent-based approach. According to the oxford dictionary, the complexity of a system involves two or more components which constitute parts of the system, and are (1) joined in such a way that is difficult to separate them, and (2) closely connected. This duality determines two dimensions of complexity, the distinction in several components leads to study an appropriate structure of the components, as the connection leads to study how they dynamically interact. Complexity is a property found in many kinds of natural systems, in physics, biology, social sciences... It is at the crossroads of different approaches, system theory, artificial life, cybernetics, artificial intelligence... Due to our local context and scientific environment, we began to tackle complexity

---

[1] IREMIA, University of La Reunion, BP 7151, 97715 St Denis Messag. Cedex 9, La Reunion, France. Tel: (+0) 02-62-93-82-84 Fax: (+0) 02-62-93-82-60 Email: marcenac@univ-reunion.fr

[2] LICEF, Tele-University, 1001 Rue Sherbrooke Est, 2eme etage, Montreal, Quebec, Canada H2X 3M4. Tel: (+514) 522-3540 Fax: (+514) 522-3608 Email: sgiroux@teluq.uquebec.ca

[3] This project is both supported by the Institute of Research in Mathematics and Computer Science of the University of La Reunion and by the Geophysics Laboratory in Grenoble (France).

through the modeling of natural systems, such as those studied in geophysics, earthquakes and volcano eruptions.
Furthermore, a complex system is seen as a non-linear system, in which the result of interactions between its components exceeds their individual contributions, and where mathematical models do not provide efficient solutions to understand and thus predict the behavior.

In such a context, the main issues of modeling are to find a way to reduce, to organize the system complexity and to describe suitable components. The challenge is then to understand why interactions between some parts of the system lead to a global behavior, even when it is not clear that there is an underlying theory. The issue is not to model some existing systems which are naturally distributed, but rather to model complex systems as distributed systems, where the behavior is the result of interactions between its parts. This point of view can be, at a first glance, associated to C. Langton's works on artificial life in Santa-Fe [29].

In this framework, traditional artificial intelligence techniques often fail when modeling such complex worlds within artificial systems: considering a complex system as a unique entity, and the behavior as the result of this uniqueness does not allow enough factors to be described and the intrinsic complexity of the system to be taken into account. Therefore, artificial systems that spew realistic behaviors require the study of independent acting variables.
Multiagent modeling helps us to investigate such an approach. The agent paradigm seems to be an appropriate framework, because it provides computational models which aim naturally at representing the phenomena locally, and in which the emergence of the global behavior is the result of interactions between agents.

This work is motivated by the following factors:

- Reducing complexity to model complex systems requires appropriate structures,
- the agent technology provides good results when the complexity of the system can not be globally expressed,
- in the framework of complex systems modeling, there are no appropriate tools (especially in simulation),
- intrinsic mechanisms of the agent-oriented approach, and more particularly the emergence of global behavior are complex, not well understood, informal or not clearly identified.

To address these issues, a layered architecture, called GEAMAS[4] is introduced. GEAMAS is seen as a multiagent development platform, intended to develop simulation applications for complex systems. Imagining a layered architecture is one of the possible answers to the complexity reduction by abstraction [16]. It allows a better understanding of the complexity through the expression of the fine-grained or coarse-grained components of the system. In addition, such an architecture can provide information on how macrobehavior could emerge out of interacting microbehaviors, and why the multiagent approach works in this context.

This architecture is based on *three abstraction levels.* Each successive level represents a higher level of abstraction. Each abstraction level describes a degree of knowledge complexity and applies a model of agents, through the expression of interaction, behavior and evolution capabilities. Each level is independent, executing processes asynchronously and representing a higher level of abstraction than the one below it. Furthermore, the architecture implements the *recursion* property to greatly reduce the design of the system, by applying the same agent-model to coarse-grain and fine-grain components. To move up and down levels, two fundamental mechanisms are introduced: *Decomposition* and *Recomposition.* Decomposition allows to transfer information to lower levels, until Recomposition allows to transfer information to higher levels.

The agent-oriented platform *ReActalk* has been chosen as a basic shell for programming the GEAMAS architecture and experimenting with agents. ReActalk is made up of successive layers built upon

---

[4] Acronym for GEneric Architecture for Modelling with Agents and Simulating.

Smalltalk-80, and supports large mechanisms of implementation for both individual agents and global systems. It provides a safe combination of passive objects (Smalltalk classes) and actors, bringing important advantages to the implementation of agents. To ease the designer's task, an agent-oriented design methodology is proposed. The methodology points out what is perceived to be the most important issues in the design of multiagent systems for simulation of complex systems: the agent's *role*.

To validate the architecture, a simulation application to help in predicting volcano eruptions has been investigated. The whole application was tested by our team, and resulted in more than one hundred simulations during the first twelve months. The interpretation of this year's results were very encouraging, and revealed some critical parameters playing a role in such eruptions.

This paper is divided as follows. Section 2 shows the contributions of distributed artificial intelligence techniques to complex systems modeling. Section 3 details the GEAMAS kernel, the three abstraction levels and the agent model. Section 4 shows how recursion helps in designing the system, and how behavior emerges from agent interactions. Section 5 presents the implementation of the architecture with ReActalk. Section 6 discusses of the agent-oriented design methodology to derive systems from the architecture and section 7 describes the example of a simulation application, gives an example of working, and draws up a report of interesting results. Finally section 8 concludes the paper by pointing out current investigations and further research perspectives.

2. AGENT AND COMPLEX SYSTEMS MODELING

2.1. What exactly is an agent?

The notion of an agent has emerged at the crossroads of distributed computing—artificial intelligence and embedded systems—and is likely to be significant in software development. Multiagent systems are, typically, distributed computational systems in which several autonomous agents interact and work together to perform tasks to satisfy a set of goals. In this sense, a multiagent system is a collection of autonomous agents, each having its own capabilities and role, related to a common environment. We should here highlight the distinction between parallel systems where global control exists and such multiagent systems where problems are solved by agents using distributed control.

The meaning of an agent is hard to define and depends on the context tackled. There is no common ontology (that is no common structured set of concepts) in agent technology. However the unifying aspect between the different points of view concerning what an agent could or should be according to J. Ferber [14] [15], is that: "an agent is a physical or potential entity, acting on itself and its environment and disposing of a partial representation of this environment. An agent pursues an individual goal, communicates with other agents, its behavior being the consequence of its competences and communications with others".

The agent concept is often compared with the concurrent object concept or actor, an active object seen as a concurrent task. An agent, as well as a concurrent object does not wait for another resource to proceed, and can perform actions through asynchronous messages passing. However, an agent is also independent (it can perform tasks without explicitly receiving the order) and *autonomous*. The autonomous part of an agent determines its self-governing: an agent is able to take an initiative and exercise a flexible degree of control through its own actions, by dynamically choosing which actions to invoke, that is not the case for concurrent objects. An agent *interacts* in a concurrent and an asynchronous way with its environment to draw up a state of the other agents, and reasons to provide the most efficient behavior. Therefore, an autonomous agent should dispose of different abilities such as

perception, action or sometimes reasoning to keep control on all these activities. Actions are then controlled according to the agent's and environment's state. For instance, let's consider a human implemented as an agent or an object: the agent may decide to open an umbrella or put sunglasses on, according of the weather, as an object should explicitly receive the order to execute.
Finally, an agent is evolving during its life, thanks to relationships kept with other agents, and *adapts* behavior to the environment. From this perspective, an agent is indissociable from its environment, and their symbiosis determines the development of both.
The agent technology traditionally distinguishes between cognitive and reactive agents. A cognitive agent has a symbolic and explicit representation of its environment, as well as reasoning capabilities. Examples of this approach can be found in [9], [13], [18] or [30]. On the contrary, a reactive agent is defined as not including any kind of advanced knowledge, nor using any complex reasoning. Reactive agents only work when responding to stimuli. Examples of this approach are described in [8], [11], [14] or [35].

2.2. Are agents adequate to model complex systems in geophysics?

Many geophysical researchers have found out common features for these phenomena: for instances, spatial and temporal scaling laws, small and slow driving perturbations, long range interactions... To try to understand complex systems such as natural phenomena, the classical approach is to aggregate data, knowledge and hypotheses to build a model of the physical world. This model[5] is generally expressed by mathematical relationships between variables, matching some real physical magnitudes, such as differential equations for instance. Simulations are then used to analyze the properties of theoretical models. Ran on computers, simulations provide tests to validate the theoretical model. Results can then be processed and exploited with the help of statistical techniques to verify the given hypotheses.

But, in this area, most classical models have failed both to understand the underlying processes and to predict future behavior. Though theoretical models have provided significant contributions in physics, they can not avoid from some negative results: first, a mathematical model links parameters which belong to the same granularity level. It is not possible to link microscopic behaviors with global variables which match macro level values [15]. Second, the expression of the complexity is poor, because a variable represents an abstract view of the reality in an equation and encapsulates behaviors by masking underlying complex parameters. Third, a mathematical model is unable to adapt the structure of the real world and microbehaviors which arise as individual behaviors performed during the simulation at a microscopic level. It does not allow to understand the system evolution and the internal factors organization.

The study of this kind of system in Geophysics has led to the concept of *Self-Organized Criticality* [2], to explain the "repeatability" of phenomena in nature, which could obviously be observed in scaling laws. Such systems are driven by highly non-linear behavior, where a small external perturbation could generate a large-scale phenomenon at a critical state of the system, but without predicting when it could appear. The best well-known example illustrating this property is the repeatability of avalanches in a sand heap, where one of the important results is the apparent robustness of the scaling laws: avalanches regularly happen while grains are randomly added during experimentation. Such a system describes a degree of complexity more important than its parts, and includes properties which can not be reduced to those of its components. This non-reduction is assigned to the presence of interactions which

[5] The word "model" is here understood in a mathematical sense. The word "model" is here synonym of agent model, in a computer science sense.

dynamically unify the system components, and from which a phenomenon appears by affecting a part of the system.

From a computational point of view, "classical" artificial intelligence techniques often fail when modeling such complex worlds. One of the major reasons for this failure is to consider a complex system as a unique entity. Thus, it does not provide enough semantics to give satisfying results. A centralized program remains too close to its reasoning, and is unaware of its environment. Tiny mechanisms can not be deeply taken into account, leading to an inadequate description of the real world. The individual actions, taking part of the elaboration and therefore the organization of the system structure are being ignored by this approach.

The agent paradigm brings a new solution to the previous issues. It seems to be an appropriate framework, because it provides computational models which naturally aim at representing local phenomena, and in which the emergence of global behaviors is the result of interactions between agents. The model[6] of agents we propose allows a representation of the real world in autonomous parts, with their interactions and evolution capabilities. Even if this point of view leads to a weak notion of what an agent is [38], it is sufficient to model complex systems for simulation needs and to help in understanding their behavior. Designing more complex agents, in terms of human-like concepts such as GRATE [28], a layered architecture in which the behavior of an agent is guided by the mental attitudes of beliefs, desires and intentions, or IRMA [6] (stands for Intelligent Resource-bound Machine Architecture) which integrates a reasoner, several analyzers and more complex processes, is not so rational in this specific case.

The next section presents the fundamentals of our framework GEAMAS, a generic architecture of agents to build simulation applications.

3. TOWARDS A THREE LEVEL ARCHITECTURE

This section exposes the problems surrounding the construction of multiagent systems that satisfy the properties specified in the simulation of complex systems. When experimenting with agents, the main issues are [26]:

- How to organize a multiagent system as a society in which intelligent behaviors will arise as a result of agent asynchronous interactions. More specifically, due to the underlying complexity of the system, we will see that traditional organizations do not give significant results in this framework.
- How to define an autonomous agent, which matches the complexity of the real world.

3.1. About multiagent architectures

A multiagent architecture is defined as one that contains a model of the real world, in which decisions and reasoning are distributed among the agents. Traditionally, multiagent architecture describes two levels, as shown in Fig. 1. below, representing a cognitive architecture.

---

[6] An agent of level *i* will be noted *i-level agent* in the rest of the paper.

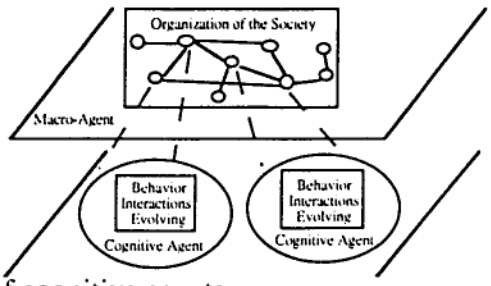

Fig. 1. Traditional architecture of cognitive agents.

This type of architecture identifies (1) macro-agents as related elements in which behavior will be observed, describing agents in a "society", and (2) interactions, microbehaviors and evolution facilities of the agent. However, in a complex systems modeling framework, two kinds of features are to be considered. On the one hand, non-predicted and complex real worlds need to split the complexity into atomic pieces to better understand, control, and isolate underlying phenomena. On the other hand, agents can not be expected to model their surroundings in details, as there are too many unknown events and parameters to consider. It is therefore not surprising that neither purely reactive, nor purely cognitive architectures are capable to judiciously manage the complexity within a system and describe sophisticated phenomena. To increase simulation features and to introduce a degree of complexity in our architecture, the GEAMAS approach is to marry cognitive and reactive architectures. Such an architecture becomes a hybrid, according to the terminology found in [38]. In this architecture, sub-multiagent systems are arranged into three levels, with the top layers dealing with information at increasing abstract levels. Each level is independent, executing processes asynchronously, and representing a higher level of abstraction than the one below it.

The third level is the lowest layer which describes reactive behaviors. The first level is the highest and deals with long-term goals whilst the second level meets intermediate goals and represents intermediate structures. This second level makes the link between the global goal of the whole system and reactive behaviors, as shown in Fig. 2.

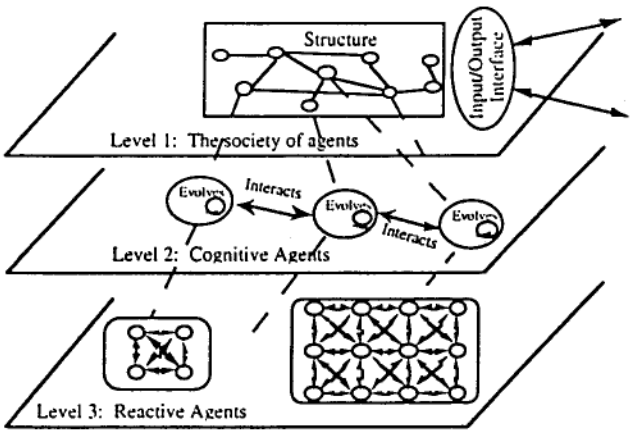

Fig. 2. Three abstraction levels to model the complexity of a system.

3.2. The first level and the society model

The first level manages agents in the system in order to match global specifications and is inscribed in a model called the *society model*. This model mainly expresses the role, the interface and the whole organization of the system.
The role of the system is seen through its external behavior, and driven by an input/output interface with the external world (software, human...). The interface also manages input and output parameters for simulation needs.
The first level also describes how agents are organized to form a society, and is then responsible for the collective of agents representing inter-agents connections (as interaction possibilities between agents). This organization is defined as the description of agents' dynamic relationships used to structure interactions within the system. It can not be confused with a hierarchical tasks view leading to a functional division, as mentioned in M. Fox's excellent paper [16], to be illustrated in human organizations or economics. This kind of hierarchical organization requires a tight coordination between agents. In natural phenomena modeling, it is quite different. No coordination could be associated in the organization, as the system behavior is not determinist, emerging from bottom to top levels, and is not driven by such centralized decision making.
GEAMAS organizes the society as a *network of acquaintances*, forming a competence network, where agents correspond to nodes and interaction possibilities (acquaintances relationships) to edges. The choice of a network as a data structure to organize agents is justified by its flexibility and its general nature to be adapted to several situations. In addition, it allows the description of topologic representation of the real world, where the geometry can be implicitly represented. Indeed, the number of acquaintances of each agent determines the topology, and constitutes a very interesting feature, showing how a real world could be designed as a three dimensional network. This parameter is called the *degree of communication* or *connectivity* of the agents in the system. As connections determine the ability of an agent to communicate, a part of the communication protocol is then defined by setting this parameter. This number of connections describes the influence an agent can have on another, and defines a spatial and geometric disposition of the agents. A simple heuristic to represent the degree of communication is to consider a "corona" in a plan, as illustrated in Fig. 3. below.

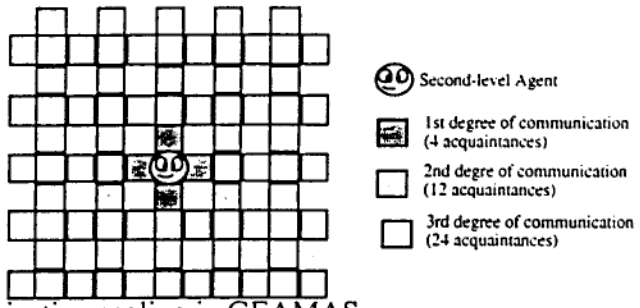

Fig. 3. Degree of communication scaling in GEAMAS.

However, modeling the agents organization in a Cartesian plan falsely induces a plane view of the real world structure. It is therefore possible to abstract this projection, when the degree of communication is greater than 1. This gives a view of the real world in three dimensions (3-D grid), when using 3-D connections in a network. A plan then becomes an abstract view of the real world, and if two agents are not immediate neighbors, they can even be linked together. Fig. 4. illustrates a third degree of communication.

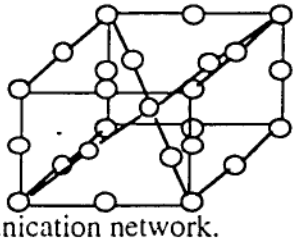

Fig. 4. A 3-D grid example of a communication network.

3.3. The second level and the agent model

The second level of the architecture describes the components of a first level agent, each one being seen as an autonomous agent. This level only describes high-level, *cognitive* agents. They will often be called *second-level agents* hereafter. Such an agent defines an abstraction gathering activities which are identified as a whole. This set of activities is the result of a joined effort of several fine-grain agents (that is third-level agents in GEAMAS). Therefore, this kind of agent meets intermediate goals, and describes more complex behaviors than those of fine-grain agents.

Each second-level agent is built from a model, called the *agent model*, and described by the triad interaction, behavior, and evolution:

- Interaction is a basic mechanism when working with agents; interaction can be defined as a dynamic relationship of at least two agents by means of a set of reciprocal actions. Interactions consist in actions exchanges between agents, from which the result of the system will emerge.
- The agent's behavior represents what the agent is able to do during its life. In GEAMAS, the agents' behavior can be internal or external. Internal behavior describes (1) the autonomous part of the agent (that is a kind of control which determines room for decision making), and (2) its independence, assuming that an agent can work without external solicitations. External behavior is provided by the agent when receiving external solicitations (such as events).
- Finally, the last property which should be carefully studied is the agent's capabilities to evolve over time. In a universe describing natural phenomena, each component is subject to non-linear modifications over time. Modifications affect both the agent's data and its behavior, and depends on what was done by the agent during its life.

Two kinds of dynamic evolution have to be considered: structural modification through the re-organization of the acquaintances network and behavior modification.
The first one constitutes a fundamental feature in complex system modeling, and is often called *self-organization*. Self-organization deals with the appearance, in a specific context and in an active environment, of new structures, not previously identified and from now on irreversible within a system of interacting entities. The appearance of such emergent structures in a component C is seen as the consequence of interactions between fine grain components of C. In addition, GEAMAS is equipped with "retroaction" capability: in some cases (earthquakes or volcano eruptions for instances), the affected part of the system is forming some new remarkable structures where components can be forced in their turn. These structures and the resulting constraint applied upon their components are playing a major role for the future behavior of the system (for instance, there is no eruption, nor earthquakes twice a time at the same place).

Therefore, this property allows a system to be simulated without any initial constraints. The network is then organized as the events allow it, until the system reaches a critical state. This feature allows a better understanding of self- criticality. In self-organized criticality, the term "self-organized" refers to the fact

that the critical state is an "attractor" for the system's dynamics [4]. The critical state is then the most favorable state to allow phenomena observation.

The behavior modification is also a very interesting feature to investigate, as it contributes to the autonomous part of an agent; this characteristic authorizes an agent to reproduce adequate behaviors, that is behaviors the best adapted to the environment and to the current situation. Second level agents are in charge of this kind of modeling: by interpreting what takes place in reactive agents, cognitive agents get information concerning the underlying processes in the third level. They can then provide specific behaviors emerging from this interpretation.
This view is very close to reality: a cognitive agent examines the behavior of its components (reactive agents), keeps track of them before deciding what to do. In the example of the sand heap described in section 2.2 before, adding a grain of sand to the heap can set off an avalanche. Grains are seen as reactive, but do not have enough knowledge to be able to describe the whole phenomena. On the opposite, a sand heap is seen as a cognitive agent: by analyzing each force applied between reactive agents, it can identify the avalanche and keep track of the weaknesses of its underlying structure. This important feature of the model will be discussed in section 4.2.

3.4. Reactive agents and third level

The third level of the architecture is composed of reactive agents which model second-level agent's components and describes linear behaviors. Thus, the complexity of a second-level agent can be understood from the study of reactive agents composing it. Reactive agents are characterized by an internal state and basic actions. Such agents will often be called *third-level agents*, *cells* or *micro-agents* hereafter.
Third-level agents are *reactive* as they generate behavior without explicit representation of the domain nor global knowledge on the system's state. They simply interact with their surroundings, and evolve over time by updating their internal state. They get close to each other, and form a compact group. They act in response to events that are too fine-grained to be understood by second-level agents. In the third level, interactions between reactive agents are provided by signals, as stimuli-reactions. Signals do not bear semantics, and their meaning depends on the interpretative ability of the receiver. J. Ferber demonstrates in [15] that the intensity of a signal emitted by an agent decreases as a function of the distance between agents, and that the microbehavior of agents is then strongly dictated by their relative position in the topological structure. In GEAMAS, microbehaviors are modeled by micro-agents, and micro-agents evolve in a world whose structure is defined by the agents' communication network (2-D grids, 3-D grids, abstract connectivity...). Their behavior is locally defined and the agent can solely interact with its immediate neighborhood. Agents' behavior and state are therefore influenced by the actions of their neighbors. When an agent acts or changes state, some of its neighbors may react, producing avalanche behaviors.
This point of view is well appropriate to natural phenomena simulation: external events, simulating constraints applied on the structure, are performed by each cell. Each cell reacts by changing state and calculating the remaining constraint to spread over the network. As the constraint intensity is decreasing when cells perform the constraint, a catastrophe is not always cause by the external event, that is the reason why the system behavior can never be determinist.

The next section shows how the agent and society models are applied at each level, and how interactions occur to provide emergent behaviors.

4. RECURSION, DECOMPOSITION AND RECOMPOSITION

4.1. Recursion

The recursion property constitutes an important feature to greatly reduce the intricacy of the system design, by applying the same agent model to coarse-grain and fine-grain components. It expresses that an agent of level i[7 8] is seen as a set of $i+1$-level agents, having the same properties. Then, a $i+1$-level agent is viewed as a sub-multiagent system, having the same properties as the global multiagent system, that is the first-level agent. Recursion unifies systems and agents: an agent can always be perceived as a multiagent system, and a multiagent system can always be viewed as a single agent. This property allows us to integrate the system in a same frame mode and to provide knowledge divisions which help to reduce a design's intricacy when tackling complex systems. An immediate advantage is the model reusability: agent and society internal structures, as well as algorithms are thus independent of the domain, and can then be reused at different levels within the architecture.

A question now arises: when tackling natural phenomena modeling, can we have more than three levels? For instance in the case of a typhoon or a tornado, recursion carries through to about three stages (typhoon, kinetic volume of air in mouvement and air molecules). After that, the structures are too small for further dissection. This characteristic is applied to the GEAMAS architecture: propagating dissection of agents modeling real-world components would make the system non realistic. A micro-agent describes the most tiny grain and can not be decomposed again without being out of the system's goals. The limit of precision is reached when the agent model can not be inscribed in the expected grain level. As long as this limit can then be set, complexity, modeled on three abstraction levels, is distributed and reduced enough so that the system behavior could be observed and understood.

4.2. Running the architecture: filtering Process and behavior emergence

During run time, all agents work in parallel, each one achieving its task. During the simulation, interactions between agents occur each time an agent needs to transfer or ask for information from its surroundings [21]. As we have pointed out before, communicating by interactions is essential for the emergence of the global behavior of the system, and two kinds of interactions can be identified through the architecture: interactions between agents belonging to the same level or interactions between agents belonging to different levels. Interactions at the same level are performed through standard asynchronous message passing, according to the agent-philosophy. This is the point of view embodied within our architecture. Interactions between agents, not belonging to the same level, show the transfer of information between an agent and its society or vice versa.

Two basic mechanisms are defined in the architecture to achieve the last type of interactions: *Recomposition*, from microbehaviors up to macrobehaviors and *Decomposition*, from macrobehaviors down to microbehaviors. This kind of interactions expresses dynamic conceptual links, transferring information between two different abstraction levels, and defines messages passing primitives which their own semantic. They are also performed by asynchronous messages, but are cut off from others due to their specific semantic contribution.
A *i-level* agent can only recompose (respectively decompose) with a *i-1-level* agent if the *i-1-level* agent is the society. This property is close to the aggregation relationship in object-based systems [3]. However, the aggregation relationship only describes a static notion, as it only represents a semantic link between an object and its *composite* objects (for instance a car owns an engine, wheels...). Decomposition and Recomposition mechanisms go further: they represent an aggregation in a dynamical sense of it. They allow needed information to be interpret by higher levels and to provide high-level

[7] In the rest of the paper, writing i assumed $i \in [1..2]$.

behaviors. In the aggregation relationship, a component can not be deleted without losing a part of its semantic; Decomposition and Recomposition work in an open environment, that is adding or deleting components does not change agent's semantic. They just express some types of message which circulates within the architecture at run time.

T. Ishida and al. [27] were first introducing a similar notion with two primitives, composition and decomposition, but in a different context and with a different meaning: composition and decomposition are seen as reorganization primitives to be used when a system (a problem solver based in a production system in the context tackled) needs to adapt itself to dynamically changing situations. Situations in this work are driven by external requests, such as a shorter time user requirement for instance. In GEAMAS, reorganization is not realized through primitives; Recomposition and Decomposition are used to carry semantics through the different grain levels of the system.
Moreover, in T. Ishida's approach, decomposition divides one agent into two, when the environment demands too much from the organization, and composition combines two agents into one when saving computing resources is required for instance. In GEAMAS, Recomposition and Decomposition do not multiply nor divide agents, and are only used to set off some specific mechanisms of the receiver. Both mechanisms are detailed in sections 4.2.1 and 4.2.2 below.

4.2.1. Recomposition
Recomposition is the "bottom-up" mechanism which is at the root of emergent behaviors of the system. It transfers information from level *i* to level *i-1*, such as from micro-agents to cognitive agents. A Recomposition message is used by micro-agents to express their *instability*. The agent's instability is expressed through the critical values of some of its parameters. These parameters define the agent's state, and are called *state parameters*. An agent is assumed to be stable when the values of its state parameters do not reach the thresholds (that is the critical value). Thresholds and state parameters are set during the design phase of the multiagent system.

The values of state parameters evolve during the simulation, based on multiple local interactions between micro-agents. When one threshold is reached, the agent is assumed to be unstable, and information is then transferred to the agent's society with a Recomposition message. For instance, let us consider an ice cube in a glass, when the temperature of a water molecule is increasing above the threshold of zero degree Celsius. In the self-organized criticality point of view, we can understand why the ice cube melts by considering that it has been informed of the temperature by its molecules. Therefore, a Recomposition message is provided by micro-agents to alert the society that something unusual is happening. Shifting from the micro-level up to the upper-level, the society collects data on microbehaviors and combines them to determine macrobehavior and then adapts itself to the situation. Recomposition then requires the society some abilities to interpret information at low-levels. A higher level behavior then emerges from such an ability. In the ice cube example, the internal state of a micro-agent is determined by its temperature which is greater than 0°C, microbehaviors are assigned to molecules' melting, and we observe the high-level behavior as the ice cube to be shrunk. In such cases, the dynamic of low-level agents governs emerging behaviors.

4.2.2. Decomposition
Decomposition allows information transfer from level *i* to level *i+1* and defines a kind of filtering process which is responsible for localizing the necessary agents, to remain consistent with the current goal of the society. When receiving a Decomposition message, the agent gets information given by its society. Two kinds of information could be addressed by such a message:

- A Decomposition message could specify that an external event is to be performed by micro-agents. For instance, the external perturbation has to be distributed between all concerned sub-level agents. This type of message helps micro-agents to be ready to perform the event. The event is thus divided

up, and "sub-events" are transferred to micro-agents which are the best able to process them. In the ice cube example, one event could be "the ice cube is plunged in a glass whose average temperature is 12°C". To understand what happens to the ice cube, such an event is distributed in each molecule lining the ice cube.

- Conversely there may be laws or knowledge on a macro-level that must be observed by micro-agents; for instance, knowledge that constrains micro-agents moves. As complete information is made available on the first level (all events performed through the input interface are first processed in the first level), the macro-agent must have authority to effect changes in the organization. A society is the right place to express such knowledge; it may constrain part of the state and behavior of their subordinate micro-agents, either by acting on the world structure or on individual micro-agents. For instance in the ice cube example, such a global constraint makes it a rule to keep the shape of the ice cube when melted.

The next section gives some details about the implementation of the architecture, which has been realized in ReActalk, a computational environment for experimenting with agents.

5. IMPLEMENTATION OF THE ARCHITECTURE

5.1. ReActalk as a basis for GEAMAS

GEAMAS has been developed with the help of ReActalk [20], a platform for reflective actors in Smalltalk-80. ReActalk can be seen as an extension of Actalk [7]. Actalk is an actor platform for the study of actor paradigms within Smalltalk-80. Indeed, Actalk is a minimal actor system built on top of Smalltalk-80, and designed in order to provide a framework for the study and the exploration of actor languages such as G. Agha [1].

Smalltalk-80 provides the structural reflection as well as the minimal object environment, and Actalk provides their asynchronous manipulation. However, as we pointed out in section 2.1, an actor is different from an agent because an actor does not take into account its system membership when an agent is necessarily indissociable of its environment. In addition, an actor neglects the autonomous part of the agent: it does not control its behavior nor allow to choose the best way to achieve a given task, because it is too linked with the class which has defined it. From these establishments, S. Giroux has developed a reflective approach of an actor to implement the notion of agent. Therefore, an agent is seen as a reflective actor, where reflection is now "operative", allowing an actor to be cut off from the class which gave birth to it [19]. Thus, this approach gives an actor autonomous and evolution capabilities. From our point of view, a reflective actor models: (1) the components of a system as objects, (2) their concurrent evolution as actors and (3) their autonomy and independence with operative reflection, which easily supports the control of macrobehaviors emergence out of microbehaviors.

For our concerns, the GEAMAS architecture models several agents interacting with one another and embedded into three grain levels, where macrobehaviors dynamically emerge from the interpretation of micro-behaviors. Operative reflection provides a natural framework for expressing the ability of an agent to interpret information coming from fine-grain agents. Recomposition, which triggers this interpretation, is isolated in a specific class implementing this particular kind of asynchronous message.

5.2. The kernel of ReActalk

The kernel of ReActalk, built from Smalltalk-80, is illustrated in Fig. 5. The inheritance mechanism was used to define both Actalk and ReActalk layers, as successive extensions of Smalltalk-80.

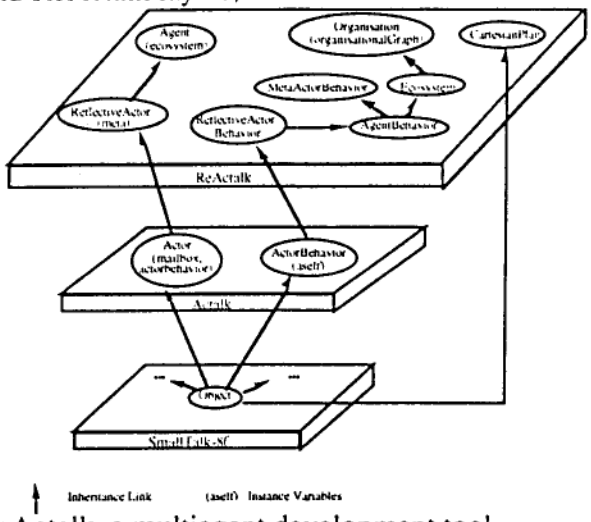

Fig. 5. Simplified view of ReActalk, a multiagent development tool.

Implementation of autonomous agents:
In Actalk, an actor is implemented by two components: a behavior part, responsible for message interpretation and an actor part responsible for message processing. Both parts are described by two Smalltalk classes: *Actor* and *ActorBehavior*. An *Actor* is composed of a mailbox, organized as a LIFO (Last-In, First-Out), which receives asynchronous messages sent to the actor, and a script which describes the actor's behavior after reception of a message.

The actor's mailbox and script are represented by two instance variables of the Actor class: *mailbox* and *actorbehavior*. Methods are also provided in the Actor class to initialize and access these two instance variables. The general actors' behavior is to process messages and, in other words, to implement the concurrent model of computation. It is described in the *ActorBehavior* class. This behavior is quite simple: after its creation, an actor consults its mailbox permanently and answers potential messages sent to it. This examination of the messages is performed by a never-ending process in the mailbox. At the ActorBehavior class level, an instance variable, called *aself*, keeps in memory which actor the behavior must be applied to.
In the ReActalk universe, the conceptual notion of an autonomous agent is driven by such actors (implemented by two classes, *Agent* and *AgentBehavior*).

Implementation of a society of agents:
Each agent moves in a global entity representing the society of agents, which forms the *Ecosystem*. The Ecosystem class naturally inherits from the AgentBehavior class, but can be considered as an abstract class, in the sense that it does not add facilities to the AgentBehavior class. It is simply a common way to properly separate an agent and its society. An ecosystem is responsible for all agents it is composed of, and is associated with a structure to organize the society, and a structure to dynamically build and manage the society.

To organize the society, a specific class, named *organization*, describing the society as a graph is derived from the ecosystem. The graph is built from the location of agents where one is compared with the other: therefore, the agent's position and the neighborhood determine a matrix in which each agent has its own collection of neighbors. The graph is accessible by an instance variable, *organisationalGraph*.

To define the structure and space in which agents evolve, ReActalk proposes a well-adapted structure for this kind of modeling: the *CartesianPlan* class. By making an instance of the CartesianPlan class explicit in the organization class, the society of agents is managed in an efficient and dynamic way. This instance has been named *universe*, and is assigned to the organisationalGraph instance variable, in the corresponding instance of the organization class, as shown in Fig. 6.

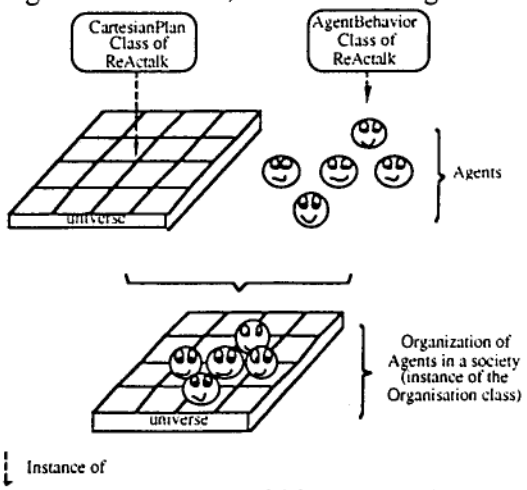

Fig. 6. Organization of agents in a Cartesian plan within ReActalk.

Reflection and meta-modeling

The adaptation facilities of an agent are based on the operative reflection capability of ReActalk, implemented by a meta-agent. In this context, the meta-agent allows an agent with keeping control of its activity and having an effect on its behavior. In the GEAMAS architecture, this kind of reflection is used to interpret the instability of low-level agents when a Recomposition message is received by a society. Reflection is implemented in ReActalk with three major classes (see Fig. 5): first, the *ReflectiveActorBehavior* class specifies reflective agent's behaviors; second, the *ReflectiveActor* class specifies which meta-agent is linked to the agent (by the *meta* instance variable). Finally, the choice of the behavior to adopt is implemented in the *MetaActorBehavior* Class, which describes how to adapt the agent's behavior according to the current state of the neighborhood, and Recomposition message arguments.

The meta-agent provides an easy access to adaptation facilities of an agent. When an agent is created, a meta-agent is attached to it, that is an instance of the *MetaActorBehavior* class is created and associated to the agent. This instance is stored in the *meta* instance variable of the corresponding *ReflectiveActor* too, and is then always accessible by the agent. This meta-agent acts as a private interpreter and is itself a society of agents thanks to reflection. Recursion is then naturally implemented by this mechanism. Fig. 7. below shows how the reflection mechanism is used in the GEAMAS context, when a cell, a second-level agent or a macro-agent is created.

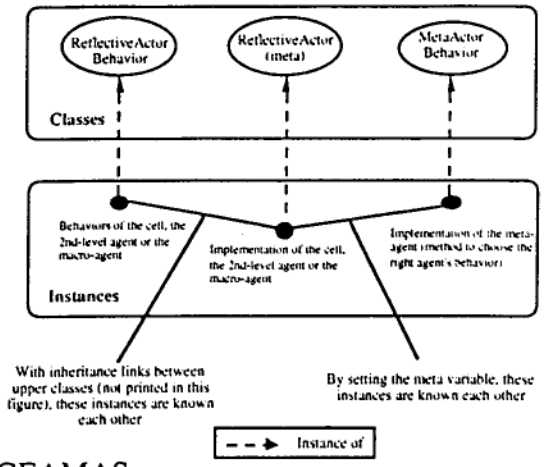

Fig. 7. Using meta-agents in GEAMAS.

The next section draws a report of what was perceived as the most important facts when designing systems with GEAMAS.

6. DESIGNING THE APPLICATION

This section tackles a major problem to solve when building agents: the translation of the real world into an accurate and adequate description to be given to the model. We first analyze some specific considerations in agent-oriented design, before proposing some hints to help designers to build applications in the GEAMAS context.

6.1. Specific considerations in agent-oriented design

From a designer's point of view, two classical approaches can be considered when building multiagent systems: bottom-up or top-down. The first one consists of looking at the application and more particularly at the different tasks it has to provide. In this kind of organization, an agent is assigned to a task and exists in a concurrent way with others. This organization needs data concerning the domain it is working with. The domain is commonly represented outside of agents, in a database or in a blackboard system [25]. In this case, the application is built by adding agents to the domain, these agents working in parallel. This approach is usually tackled when software is already implemented, and when new extensions are needed. This type of extension of a single system to a multiagent system is a feature of open systems [26]. Due to existing constraints, this the case in most industrial projects. The software design method is then *bottom-up* oriented, as shown in Fig. 8:

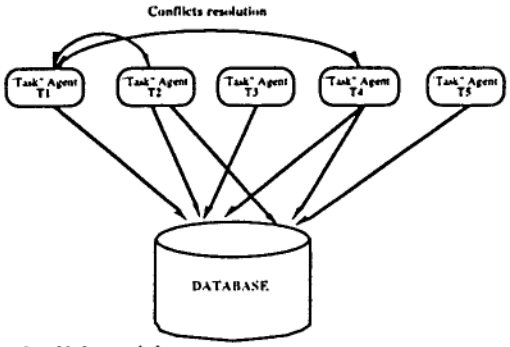

Fig. 8. A bottom-up approach to build multiagent systems.

This kind of work provides good results when existing software is not able to deal with new features such as concurrence or constraint satisfaction optimization. Examples of such architecture are related in [37], where this method has been used to assist a user who wishes to visit someone in a complex business organization.
The second approach to directly build a multiagent system is to tackle the problem with a top-down design method. This assumes that the real world is not necessarily represented in an existing system. In this approach, the real world is cut into tiny and independent pieces, each one playing a role in the application. Internal behaviors describing the autonomous life and interactions are modeled at the same time, and are encapsulated together in agents. Our method of conceptual design of autonomous agents follows this idea; it adopts an external perspective from which agent contents can be looked at. This relates to L. Gasser's ideas [17] where the agents' organization is treated as aggregated local components.

6.2. Designing simulation applications with GEAMAS

The architecture of GEAMAS describes an abstracted model of the real world. This foundation is inherent to the problem to be solved, namely modeling and reducing complexity. When using object-oriented analysis methods such as Coad and Yourdon [10] or OMT [36], the first phase is to understand the real world, by looking at "business objects", which is particularly difficult when evolving in an unfamiliar environment. Designing with agents does not defy the law: the first step is to look at the real world to find software components which have enough characteristics to be agents in the system and determines the logical aspect of the system, according to external specifications.
In GEAMAS, the designing task is quite well-structured, because the designer drives through a three-level architecture, and a model for both the agent and the society is provided. The designer has to match the model for the three levels.

To find the adequate separation, the methodology we propose is based on the distribution of *roles* in the system [33]. In GEAMAS, each agent represents a role and bears semantics on what it has to do for the whole system. Each role played by the agent in the application is described by a set of internal and external characteristics. These characteristics define the agent model, expressed by internal knowledge and behaviors, interaction processes, and evolution facilities of the agent. In addition, GEAMAS needs to look for the *taskability*, which expresses at what level the activities and functions performed by an agent should be described. This determines the right level of complexity and granularity. Finding the limit of complexity to be introduced is the main issue when modeling complex systems with GEAMAS.

To address taskability, three steps should be taken:

(1) First of all, begin to build the third level (reactive agents). This level inscribes only the agent model (no lower level, so no society to describe). Behaviors are quite simple, and can easily be split up into tiny pieces. Since a third-level agent can not be decomposed again, it describes the most tiny grain. The right level of granularity is reached when the designer can not describe in any more detail the agent, following the triad "interaction, behavior and evolution". If the model is not matched, a software component can not be considered as agent, even in a reactive universe. At this step, the stability of a reactive agent also has to be determined. Since the stability is given by some parameter thresholds: such an agent is stable when the values of associated parameters do not reach the thresholds (when the temperature of water molecules falls down 0°C in the ice cube example presented in section 4.2). These values will then be computed, according to messages handled by the cell during the simulation. The computation depends on the complexity of the network described in the organization, and the state of neighboring agents.

(2) The second step is to design the first level. As the third level models microbehaviors, the first-level agent models macrobehaviors. Macrobehaviors provide global results of the system, and are

embedded in a macro-agent. A macro-agent models the society as described in section 3.2 with input and output interfaces specifications. Interface specifications must answer the following: what are the input parameters of the system? What should be done with global results provided by the system?

(3) Finally, the last step is to design the second level. Two remarks could help the designer at this step: first, a second-level agent should apply the agent model for itself, and the society model for third-level agents; second, a second-level agent is concerned by a modification of a reactive agent. When a reactive agent is not stable enough, its society is alerted (that is the corresponding second-level agent), using the Recomposition mechanism. The second-level agent must then respond to the instability of this reactive agent, and by derivation, dynamically produces results. Therefore, emerging behaviors in the second level are directly expressed through those of cells, and indirectly expressed by the underlying structure of the cells describing the system. This point of view enforces the necessity of finding the most efficient abstraction when designing this level, which is totally dependent of the context tackled.

The next section illustrates an example of an application designed in such a way, and implemented in the GEAMAS context. The chosen application deals with the predicting of natural phenomena, and has been called GEOMAS[9]. However, all principles illustrated in this example can be abstracted from the expertise, and be reused elsewhere. A simple example, showing how messages are exchanged within the GEAMAS architecture, is also presented. Finally, some important results of the simulations are discussed, validating the approach and the architecture.

7. AN EXPERIMENTATION FRAMEWORK: THE GEOMAS SYSTEM

7.1. Using a multiagent approach to model volcano activity

The general framework of the experimentation is to provide an *agent-oriented simulation environment* as a test-bed for complex modeling of geophysical mechanisms. The example chosen to model such mechanisms deals with volcanic eruptions of the Piton de La Fournaise, in La Reunion Island.

As is the case with any complex system, the Piton de La Fournaise volcano is characterized by a very complex structure, which does not adhere to any known physical law. In situ observations argue that the dynamic of the volcano is driven by non-linear processes and by small and slow perturbations. We have seen that on such a basis, the volcano is considered a self-organized critical system [24]. The scale invariance is characterized by the repetition of watching behaviors during time in spatial and temporal scaling laws. This property has been revealed by numerous data surveys on Piton de la Fournaise, for instance, eruption size distribution over the last seventy years [23]. When focusing on eruption size distribution, data have then been noted as power-law representations, viewed as two segment lines (see Fig. 9).

[9] Acronym for GEOphysics and MultiAgent Systems.

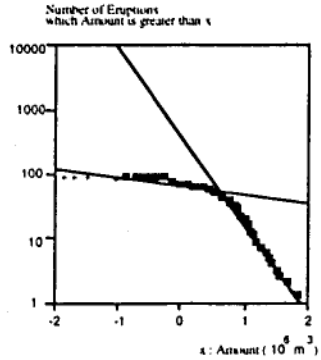

Fig. 9. Number of eruptions according to size, occurring between 1920 and 1992.

The behavior of the volcano is completely determinist, because it follows a well-determined law, but at the same time, is driven by non predicable behavior. From self-organized criticality, we deduce that the volcano is then seen as a global system in which underlying dynamic processes (that is energy potentials) are distributed. An eruption is not considered as the result of a single unilateral thrust, but rather as a series of interactions between components at different scales.

To better understand the mechanics behind this self-organized critical system and point out some parameters playing a role in eruptions, we designed and implemented GEOMAS, a simulation platform based on GEAMAS principles. We used GEOMAS to simulate models based on statistical physics and to explore their relevance to the complexity of the eruptive behavior of the Piton de la Fournaise volcano. The issue was to establish if the spatial and temporal complexity of natural phenomena emerges either from geometrical and material heterogeneity, or from the chaotic behavior related to underlying non-linear equations. In such a context, studying local and distributed internal processes by simulation to understand how eruptions occur might constitute a significant contribution. As the complexity of the volcano chaos has not yet been identified, the aim of the simulations is to discover some relevant variables.

7.2. Designing GEOMAS

A volcano can be seen as a network of "magma feeders", often called "magma *lenses*". Magma lenses are inter-connected continuously or temporarily, and are separated by a matrix of *rocks* coming from previous eruptions. However, one can note that magma is produced selectively; this reinforces the independent nature of the system, because it generally occurs through autonomous regulation.
Modeling a volcano has given rise to a number of challenges. First, lenses are considered as cut off from the volcano, as they have their own behaviors: according to their size, shape, magma and rock collecting capacity, they will react to internal or external perturbations. External perturbations are due to overpressure coming from elsewhere in the volcano. Internal perturbations are due to magma crystallization inside the lens. When magma crystallizes, lenses become cool, causing overpressure in the center. This overpressure can lead to isolated volcanic eruptions and a lens must then have the ability to increase its internal pressure alone. So, a lens which has just been drained has to quickly modify its state to keep its pressure at the right value and ready for disturbance arrivals. As a new disturbance appears, the lens will then be able to modify and adjust its behavior according to this new computed value, as observed in the real world. This local behavior is totally independent of what the system will provide in output, that is an eruption occurs or does not occur.

Second, the nature of rock complexity is expressed by the representation of the resistance to magma pressure and its evolution over time. This behavior should be represented as a dynamic process, as magma cuts paths through rocks during the eruption process. Furthermore, after the eruption stops, these paths form locally cooled magma pipes, called "*dykes*". A dyke is a rigid structure, therefore magma has little chance of breaking through again.

Designing third-level agents:
Microbehaviors encapsulating complexity were defined and assigned to micro-agents. The focus was put on the circulation of magma through rocks by a running fluid flow simulation (magma injection). As the real world is assumed to be made of magma lenses and rocks, micro-agents here define *cells* of both rock and magma. Lens cells control the volumes of magma exchanged through rock cells by adjusting their internal pressure. Interactions of such a cell consist in propagating pressure to its surroundings and moving magma, when its internal volume and pressure are going beyond a set limit given by a threshold. To recompute the internal pressure, a number of local rules are set by the designer before the simulation begins. For instance, in GEOMAS, three back pressure laws can be drawn:

- *constant:* when constant, the pressure is recomputed by subtracting a percent from the old value,
- *linear:* when linear, the pressure is recomputed by subtracting a random number from the old value,
- *gaussian:* and when gaussian, the back pressure law is expressed by a gaussian law.

In addition, such a lens cell cools down with time. This is represented by a "cooling law", where the temperature is recomputed at each time unit. During the simulation, if the temperature reaches a critical threshold, the corresponding society will be alerted with a Recomposition message.

While lens cells control the volumes exchanged, rock cells direct magma exchanges. During the simulation, if a threshold is reached, the excess magma is transferred to neighboring cells, sometimes generating avalanche behavior. As internal pressure in a lens cell increases, the cell expels lava, cutting paths through rocks. Each rock cell is then seen as a discrete item of space, modeling resistance to magma pressure. For instance, after magma breaks rocks to transfer its contents and be cooled, a dyke appears. A dyke is modeled by high values of resistance of some cells which set its length. Thus, fluid exchanges depend on the geometric structure of the network. At the beginning of the simulation, values of resistance and pressure are randomly chosen between two given limits.

First-level design:
As regards with our immediate concerns, macrobehaviors are expressed in terms of eruption frequency and volume. System inputs model magma injections (controlled at run time by lower-level agents), while outputs measure the amount of magma ejected from an eruption. A specific interface, discussed in section 7.4, has been designed to analyze these results.
Input and output parameters have to be chosen before the simulation begins. An example of input parameter might be the choice of the injection mode of each lens, which is a global parameter of the system. The injection mode can be continuous or discrete. In continuous mode, magma can be injected into a lens even if the network is not stable. In discrete mode, the network has to be stabilized once before a new injection can be made. The network is considered as stable when performing simultaneous "injection messages" does not involve the internal state of individual agents, that is, does not increase internal parameter values significantly. An example of output parameter might be the way to collect results. Two ways of collecting eruptions can be envisaged when an injection is applied as input in a lens: counting the amount of ejected magma from all eruptions or counting one eruption and the amount of magma it ejects.

Some other parameters are important too: the degree of communication, determining the size and the topology of the network, sets the number of lenses and rocks for a specific simulation. In addition, the

network can be initially chosen as either loaded or unloaded. When loaded, each initial lens pressure is triggered by a value which is just under that of the rock resistance. This assumes that the volcano is in a critical state and an eruption is imminent. The last parameter of the society is to choose, for each simulation, the lens to be injected and the lens able to eject magma, that is to generate an eruption. These kinds of parameters are important, because they reflect the number of constraints applied to the edifice; moreover, analysis of certain GEOMAS results shows better results with less constraints (see section 7.4 for a more complete analysis of the results).

Second-level design:
Micro-agents (cells) are aggregated into second-level agents. Two agents, lens and rock, are needed to describe the society as a group of cells.

Concerning the lens agent, internal perturbations are described by a specific cooling law: when cooling down, the internal pressure of a lens is increased, without external solicitation. This important fact establishes that a lens is able to eject magma on its own. When different cooling conditions are combined, a lens provides an isolated eruption. The issue is then to model this emerging behavior, without disturbing the rest of the simulation. Optimal conditions for isolated eruptions will be given by reactive agents modeling the lens sides. When some parameter thresholds are reached, the cell will alert its society by a Recomposition message. Information hold in this message will then be interpreted and analyzed by the lens, giving the result.
External perturbations are caused by specific behaviors: to simulate external fluid feeding, for instance from the earth's core, magma volumes are injected within specific lenses, selected by the user as input parameters of the simulation. The frequency and rate of these external injections are pre-set. When drained, a lens distributes the external injection to its cells thanks to a Decomposition message. This event is divided in order to model the fluid flow here: during the simulation, if a volume of magma V is transferred to a lens, the agent discretizes the input volume in multiple ΔVs, and injects an ΔV amount of magma in each cell. Hereafter, a low volume item ΔV will induce a modification of the local pressure in each cell. The cell has to modify its state rapidly, in order to maintain the correct pressure value and be ready for the arrival of new disturbances, as observed in real life. At the next ΔV to be injected, the cell is then ready to process the new computation. It is worth noting that this local behavior is totally independent from the global behavior the system will provide in output, that is if an eruption may or may not occur. From a programming point of view, ΔVs represent time units required to recompute the internal pressure between two messages.

The behavior of a rock agent is more simple and is expressed by its resistance to magma pressures. Interactions of a rock cell consist in propagating pressures to its neighboring components. However, when even magma penetrates the rock, it breaks the rock structure. The internal structure of the rock agent is then continuously modified. To balance this behavior, the value of the resistance is distributed through the rock cells, and is randomly chosen between two given limits before the simulation begins.

7.3. A simple example of working

This section illustrates how information is processed by the GEOMAS system during a simulation and how asynchronous messages are sent through the three levels, when an event occurs. To make important features only stand out, a simplified structure of the system is described.
The first level is represented by a macro-agent named *PitonFournaise*. This macro-agent is responsible for providing global results, that is measuring the total amount of magma produced by the simulation. Some input parameters manage the whole structure, such as the number of second-level agents (e.g. 2,500), or the law giving the distribution of magma in rocks (e.g. random distribution).

The second level is composed of agents modeling rocks and lenses components of the real world. The goal of a lens is to eject magma when possible, as the goal of a rock is to hold on to it. In this simulation, a rock agent is assumed to have a side composed of 10 rock cells, and a center of 2.
The third level describes reactive agents, that is cells. A lens cell has a double behavior: first, if a volume V of magma is injected, an ΔV will be performed as an external disturbance on each side cell, owing to a Decomposition message. The cell then computes the new value of the back pressure, according to the global law of the society, which has been pre-set by the user for the whole simulation; second, when no external perturbation is provided, a side cell cools down by decreasing the internal temperature. When the temperature is low enough, a Recomposition message is sent to the society, which has to process it, that is provoke an isolated eruption. In the example illustrated in Fig. 10 below, behaviors of both lens cells and lens second-level agents are ignored, in order to simplify the understanding of the underlying mechanisms.
Rock cell behavior is quite simple: to know how to propagate information to the neighborhood, the volume of magma received is compared with the breaking point in each agent. Side cells will be called Cs1, Cs2, ... Cs10, and middle ones will be called Cm1 and Cm2. The stability of a side cell is expressed by a resistance threshold. Reaching the threshold means that a rock does not resist the magma pressure. The cell then sends a Recomposition message to the rock society. When processing this message, the second-level agent propagates magma to surrounding lenses.

To understand how information is processed, let us look at the following example: the lens L1 in the second level breaks up, propagating a volume of magma to the neighboring rocks:

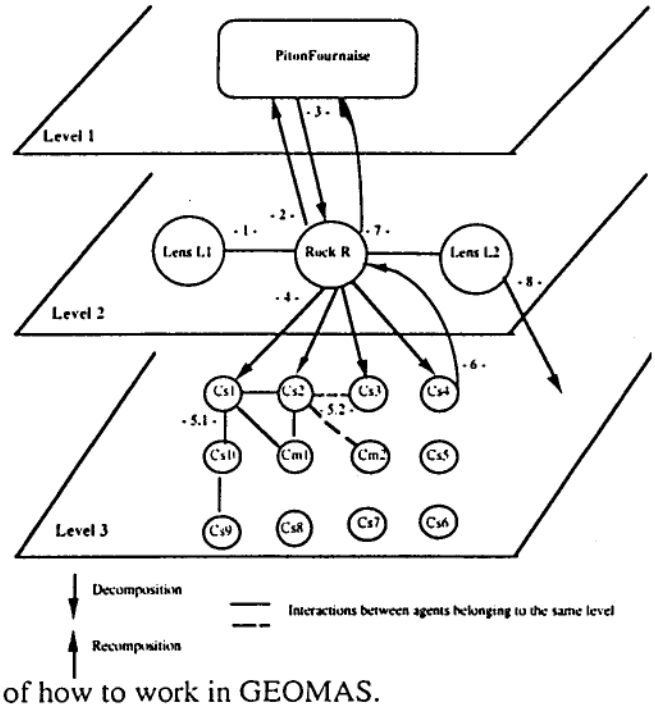

Fig. 10. An easy example of how to work in GEOMAS.

- 1 - The lens agent L1 injects magma into all of its surroundings. Note that all neighboring agents of the lens will receive this message, and will perform it in parallel. To simplify the example, we only take into account the behavior of the rock R when receiving the message.
- 2 - Agent R calls for the law giving the magma distribution in rocks to its society, PitonFournaise.
- 3 - PitonFournaise answers by giving the law and R can then calculate the magma unit (ΔV) to distribute to the rock cells.

- 4 - R sends ΔVs to the cells modeling the rock sides in the third level. This is done by a Decomposition message.
- 5 - Cs1, Cs2, Cs3 and Cs4 receive the message and process it in parallel (these cells are randomly chosen by R between all of the side rock cells).
  - 5.1 - Lets assume here, that Cs1 can not contain the magma. Cs1 therefore spreads the magma to its surroundings (Cs2, Cs10, Cm1). Lets look at what happens in Cs2 for instance.
  - 5.2 - As there already is magma in Cs2, this new magma arrival will increase its volume, and so on in Cs3... This flow of messages is illustrated by doted lines (---) in Fig. 10.
- 6 - If one of the rock cells reaches the threshold (Cs4 for instance), the society (second-level agent R) is informed by a Recomposition message type. This means that the structure of the rock is breaking. Each Cs1...Cs10, Cm1 and Cm2 cell keeps track of this breaking point by setting its internal parameters.
- 7 - The previous Recomposition message is interpreted by the society. This message says that R is breaking, and asks for magma to be transferred to all neighboring lenses immediately. In Fig. 10 however, and to simplify things, only lens L2 is shown receiving the message.
  Thus, the lens behavior emerges from interactions between cells in the third level, and from the Recomposition message, which alerts the society that something unusual is happening. While the magma is being dispersed within the volcano, the society (PitonFournaise) also gets the Recomposition message. Hereafter, the complexity of the new structure will be hidden in R, and a trace of the breaking point will have been recorded in the rock cells.
- 8 - L2, in the second level distributes the message to its cells in the third level (not illustrated in Fig. 10), and so on...

This simple example illustrates how messages are sent through the architecture and how behaviors can emerge from interactions between cells. During simulation,

(1) Going from a microbehavior up to a macrobehavior:

- eruptions are associated with specific lens micro-agents,
- a macro-agent possesses data on the geometric structure of the simulated volcano and on the overall population of involved micro-agents.

(2) Going from a macrobehavior down to a microbehavior, a macro-agent:

- controls external magma injections,
- distributes pressure,
- ensures that some physical laws are globally enforced,
- and delineates reservoirs.

7.4. Simulations and certain results

Our team of geophysical researchers has been working on these simulations for more than twelve months. This project involves a large collection of locally defined micro-agents, e.g. 50x50 lenses network, embedded within a 5,000 rocks network.

Each simulation was run on SUN SPARC Solaris 2, for 8 to 10 hours. Around 33,000 eruptions were accounted for in each simulation (only 76 real eruptions were actually registered since these observations began!). As 76 eruptions are actually registered on the Piton de la Fournaise, we chose the first 76 accounted during each simulation. The distribution of the amount of magma was then compared with the one of the 76 real data. For further analysis, a graph plotter was developed in Smalltalk-80 and

interfaced with GEOMAS outputs. For each simulation, the plotter has printed some graphs, analyzing chosen parameters and results:

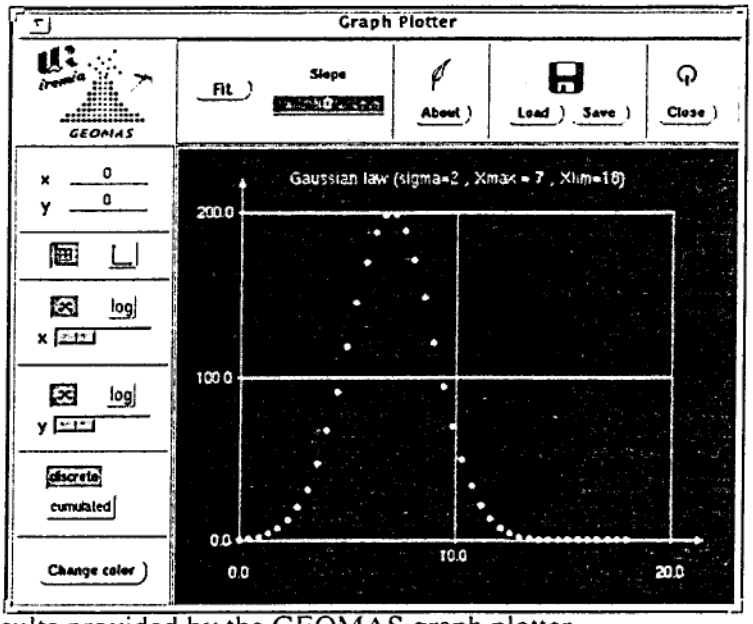

Fig. 11. Simulation results provided by the GEOMAS graph plotter.

At the beginning we had a 2-D conceptual model, where each agent communicated with its four closest spatial neighbors, but later this number was extended to 40. For large connectivity (>8), Euclidean geometry becomes useless. Although each micro-agent only has local action and is ruled by very simple laws, we were able to create interesting global dynamics, and in-situ observations were remarkably reproduced. Our model investigated processes that can neither be measured on volcanoes nor be modeled using classical linear mechanics. Three important results were found:

- The complex behavior of the Piton de La Fournaise volcano can be explained using surprisingly few identical and simple local rules. For instance, the correlation between simulation results and observations is shown for a uniform distribution of rock-cells resistance. Indeed, simulations never show similitude with real data when rock-cells resistance is heterogeneous. This emphasizes the fact that the rock is a set of homogeneous structure.
- We studied the 1920-1992 Piton de La Fournaise eruptive patterns using a network of homogeneous lens agents without a specific reservoir size. The behavior of this self-organized critical system can be maintained either by giving lens sizes a fractal or homogeneous distribution. A network of uniform lens sizes distribution appears as the most convenient parameter to reproduce the volcano eruptive patterns. This provides a first quantitative information on the non-existence of a only one magma chamber.
- An other interesting parameter was emphasized by some simulations, showing that good results can be obtained by increasing the network connectivity while decreasing the number of agents. This implies that the connectivity can not be considered as an important parameter playing a role in volcano dynamics.

These results have far-reaching consequences in the understanding of volcanic macroscopic behaviors, because they validate the hypothesis of a self-organized critical volcano, working as a network of several lens inter-connected, that was never be demonstrated before. Our assumption, based on the evidence that the volcano can be considered as a complex system of interacting entities, has been proved. Agent-

oriented simulations show interesting behavior in the sense that they capture, for specific input parameters, the basic processes observed on the volcano observed during the 1920-1992 period. In a more general way, the global behavior of a complex system (such as a volcano) has been reproduced from a model of very simple behavior, allowing to capture several basic information on volcano dynamics, as well as features that should be common to other volcanoes.

This kind of information was unattainable up to there with more classical approaches. The GEAMAS architecture then validates the agent-based approach for complex system modeling and simulation.

8. CONCLUSION

This paper has presented GEAMAS, a specific *architecture* to model and master the complexity of natural phenomena. The architecture of GEAMAS is based on *three levels of abstraction*, allowing the desired grain in each level to be described. Each level inscribes a *model* for both the society and the agent. The society is described as the connectivity of the underlying agent acquaintances (degree of communication), and input and output parameters. Agents are designed in terms of behavior, interaction and evolution capabilities. Cognitive agents inhabit the first and second level, and describe sub-multiagent systems in their underlying sub-level. An associated mechanism, *Decomposition*, distributes events on lower-level agents. Third-level agents are reactive agents, describing fine-grained behaviors. They allow second-level agents to be abstracted from the complex dynamic of their interactions. The complexity is then distributed within reactive agents, and their local interactions at run time provoke the emergence of global behaviors. Global behaviors emerge by *Recomposition*, from bottom to top levels.

This generic platform is used to develop simulation applications with multiagent systems. In GEAMAS, protocols were isolated and abstracted from the application. The software complexity can easily be increased at each stage by adding more complex protocols or developing more complex software. Thus, this approach supports an incremental development and an evolutionary design process, in which each step consists of expanding a previous version of an operational system. This platform was proved to be very helpful in understanding and analyzing the behavior of the Piton de la Fournaise volcano, Reunion Island. Other applications of the architecture, described in [31] and [32] were carried out in intelligent tutoring systems and earthquake prediction. We are convinced that this platform is appropriate for the simulation of other natural systems exhibiting similar behaviors and we actually experiment GEAMAS with social behavior analysis.

Although the systems validated in the GEAMAS context are quite small, they required careful designing and long computing time. Faced with these issues, two major research areas are currently being investigated:

-1- During simulations, agents usually face the same kinds of conditions. The repetition of the same actions leads the system to become less powerful. To address this issue, the system has to analyze the conditions which drive the best actions leading to emerging behaviors [12]. It is therefore necessary for multiagent systems to be equipped with the ability to learn, that is to automatically improve their future performances. Machine Learning techniques are good candidates for making the system more intelligent and drive the emergence process more efficiently. Because it is possible to compare a situation to a similar one, rules can be drawn allowing an agent to choose the best adapted behavior without waiting for a Recomposition message telling it what happens. By derivation, if such rules have been learnt by a cognitive agent, some input events could also be intercepted and locally performed using the same mechanism. This could avoid the need for Decomposition messages which are actually addressed to reactive agents, and therefore cut out low-level computation. Primary tests with these techniques were first introduced in the GEAMAS architecture to look for the convergence of certain factors while agents broadcast messages everywhere in the network [34].

However, learning in multiagent systems is more difficult than in "classical" artificial intelligence systems, mainly because collective behavior has to be taken into account. Learning must be distributed among agents and implemented with the same mechanism as the agent itself. The GEAMAS platform, developed on top of ReActalk, has been studied with the aim of keeping researchers close to these goals. In ReActalk, reflection could provide a powerful mechanism to easily implement emerging behaviors from observation of messages leading to the best actions necessary.

-2- As emerging behaviors are actually performed from the study of instability of reactive agents expressed through Recomposition, and the complexity grain is expressed through Decomposition, designing a simulation requires specific considerations [22]. Indeed, the Decomposition mechanism involves finding the right level of complexity grain in each level, and Recomposition involves the ability to express the emergence of behaviors for each sub-multiagent system. In such a context, complete development methodology and tool seem to be an urgent requirement. This part of the work can also be generalized in a larger context: in the simulation framework, how an agent should be conceived? what the fundamental characteristics of multiagent software platforms are in order to develop such applications? All of these questions become open issues for few years [5]. In the same way, the assignment of the multiagent approach to classes of problems can be extracted from the volcano experience. This is actually pointed out by the community as a major development in distributed software engineering.

ACKNOWLEDGMENTS

We are grateful to those people that read and commented on earlier drafts of this article, and in particular to W. Birnie, M.L. Charton, V. Dean, and to people who have brought their contribution to this work. We thank M.L. Aimelet, S. Calderoni, J.R. Grasso, D. Grosser, F. Lahaie and E. Valcares for their encouragement, enthusiasm and precious feedback. We would also like to thank the referees for their very hepful comments on various aspects of this work.

BIBLIOGRAPHY

[1] G. Agha, Actors: A Model of Concurrent Computation in Distributed Systems, MIT Press: Cambridge, MA, 1986.

[2] P. Bak, C. Tang and K. Wiesenfeld, "Self-Organized criticality: An explanation of 1/f Noise", Physical review Letter, vol. 59-4, pp. 381-384, 1987.

[3] E. Blake and S. Cook, "On Including Part Hierarchies in Object-Oriented Languages, with an Implementation in Smalltalk", in Proc. ECOOP'87, Special Issue of BIGRE, vol. 54, Paris, France, June 15-17, 1987, pp. 45-54.

[4] E. Bonabeau, P. Bourgine, "Artificial Life as it could be" World Futures, vol. 40, pp. 227-249, 1994.

[5] A.H. Bond and L. Gasser, "An analysis of Problems and Research in DAI", in Proc. Readings in Distributed Artificial Intelligence, Bond & Gasser eds, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988, pp. 3-36.

[6] M.E. Bratman, D.J. Israel and M.E. Pollack, "Plans and resource-bounded pratical reasoning", Computational Intelligence, vol. 4, pp. 339-355, 1988.

[7] J.P. Briot, "Actalk: a testbed for classifying and designing actor languages in the smalltalk-80 environment", in Proc. European Conference on Object-Oriented Programming, Cambridge, England, 1989.

[8] R.A. Brooks, "A Robot that Walks: Emergent Behaviors From a Carefully Evolved Network", Massachusetts Institute of Technology, Cambridge, MA, A.I Memo. 1091, 1989.

[9] C. Castelfranchi, "Guarantees for autonomy in cognitive agent architecture", in Proc. European Conference on Artif. Intell., Workshop on Agent Theories, Architectures, and Languages, M. Wooldridge and N.R. Jennings eds, Springer-Verlag, LNAI, vol. 890, Heidelberg, Germany, 1994, pp. 1-39.

[10] P. Coad and E. Yourdon, Object-Oriented Analysis, Object-Oriented Design, Yourdon Press Computing Series, Prentice Hall: Englewood Cliffs, NJ, 1991.

[11] Y. Demazeau and J.P. Müller, "From reactive to intentional agents", in Proc. Decentralized Artificial Intelligence, vol. 2, Y. Demazeau and J.P. Müller eds, Elsevier North-Holland, 1991, pp. 3-14.

[12] A. Drogoul and J. Ferber, "Multi-agent simulation as a tool for studying emergent processes in societies", in Proc. Simulating Societies: the computer simulation of social phenomena, N. Gilbert and J. Doran eds, North-Holland, 1994.

[13] E.H. Durfee, V.R. Lesser and D.D. Corkill, "Coherent Cooperation Among Communicating Problem Solvers", in Proc. Readings in Distributed Artificial Intelligence, Bond & Gasser eds, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988, pp. 268-284.

[14] J. Ferber, "Reactive Distributed Artificial Intelligence: Principles and Applications", in Proc. Fundamentals of Distributed Artificial Intelligence, N. Jennings eds, North-Holland, 1994.

[15] J. Ferber, "Cooperation Strategies in Collective Intelligence", in Proc. of the 7th Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindoven, The Netherlands, January 22-25, 1996.

[16] M.S. Fox, "An Organizational View of Distributed Systems", in Proc. Readings in Distributed Artificial Intelligence, Bond & Gasser eds, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988, pp. 140-150.

[17] L. Gasser, "Conceptual Modeling in Distributed Artificial Intelligence", Journal of the Japanese Society for Artificial Intelligence, vol. 5-4, July 1990.

[18] L. Gasser and M.N. Huhns, "An analysis of research in DAI", in Proc. Readings in Distributed Artificial Intelligence, Bond & Gasser eds, vol. 2, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988, pp. 3-35.

[19] S. Giroux, Agent et Acteurs : une nécessaire unité, Ph-D University of Montreal, Canada, March 1993.

[20] S. Giroux, "Open Reflective Agents", in Intelligent Agents vol. II, Agent Theories, Architectures, and Languages, M. Wooldridge, J.P. Müller and M. Tambe eds, Springer-Verlag, LNAI, vol. 1037, Montreal, Canada, 1996, pp. 315-330.

[21] S. Giroux, P. Marcenac, S. Calderoni, D. Grosser and J.R. Grasso, "A report of a case-study with agents in simulation", in Proc. International Conference on Pratical Applications of Intelligent Agents and MultiAgent Technology, London, UK, April 22-24, 1996, pp. 295-310.

[22] S. Giroux, P. Marcenac, J. Quinqueton and J.R. Grasso, "Modelling and Simulating Self-Organized Critical Systems", in Proc. 10th Annual European Simulation Multiconference, Budapest, Hungary, June 1996, pp. 1072-1076.

[23] J.R. Grasso and P. Bachelery, "Hierarchical organization as a diagnostic approach to volcano mechanics: Validation on Piton de la Fournaise", Geophysical research Letters, 1996.

[24] J.R. Grasso, S. Giroux and P. Marcenac, "A MultiAgent Approach for Volcano Behavior Simulation", in Proc. Application of Artificial Intelligence Computing in Geophysics, Boulder, CO, July 1995.

[25] B. Hayes-Roth, "A Blackboard architecture for control", Artificial Intelligence, Vol. 26, pp. 251-321, 1985.

[26] M.N. Huhns, foreword of "Multiagent Systems, a theorical framework for intentions, Know-How, and Communications", by M.P. Singh, Springer-Verlag, LNAI, vol. 799, 1994, pp. I-XII.

[27] T. Ishida, L. Gasser and M. Yokoo, "Organization Self-Design of Distributed Production Systems", IEEE Transactions on Knowledge and Data Engineering vol. 4-2, pp. 123-134, 1992.

[28] N.R. Jennings, "Specification and implementation of a belief desire joint-intention architecture for collaborative solving problems", Journal of Intelligent and Cooperative Information Systems, vol. 2-3, pp. 289-318, 1993.

[29] C.B. Langton , "Computation at the edge of chaos: phase transition and emergent computation", Physica D, vol. 42, pp. 12-37, 1990.

[30] S. Leman, P. Marcenac, M. Aube and A. Senteni, "MultiAgent Models for Cryptarithmetic Problem Solving", in Proc. Canadian Workshop on Distributed Artificial Intelligence, Banff, Alberta, Canada, May 1994.

[31] S. Leman, P. Marcenac and S. Giroux, "A Generic Architecture for ITS based on a MultiAgent Model", in Proc. International Conf. on Intelligent Tutoring Systems, Montreal, Canada, Springer-Verlag, vol. 1060, June 1996, pp. 75-83.

[32] P. Marcenac and S. Calderoni, "Self-organization in agent-based simulation", Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'97, Ronneby, Sweden, May 1997, to be printed.

[33] P. Marcenac, S. Giroux and J.R. Grasso, "Designing and Implementing Complex Systems with Agents", in I. Jelly, I. Gorton and P. Croll eds, Chapman & Hall, March 1996, pp. 27-38.

[34] P. Marcenac, S. Leman and S. Giroux, "Cooperation and Conflicts Resolution in MultiAgent Systems", in Proc. 34th ACM SouthEast Conference, Tuskegee, AL, K.H. Chang and J.H. Cross eds, April 17-19, 1996, pp. 289-291.

[35] J.P. Müller, M. Pischel and M. Thiel, "Modeling Reactive Behavior in vertically Layered Agent Architectures", in Proc. European Conference on Artif. Intell., Workshop on Agent Theories, Architectures, and Languages, M. Wooldridge and N.R. Jennings eds, Springer-Verlag, LNAI, vol. 890, Heidelberg, Germany, 1994, pp. 261-276.

[36] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, Object-Oriented Modeling and Design, Prentice Hall Int. Eds: London, England, 1991.

[37] K. Sycara, "The present and future of DAI: a study of enterprise integration", in Proc. 7th Australian Joint Conference on Artificial Intelligence, Armidale, Australia, November 1994, pp. 1-12.

[38] M. Wooldridge and N.R. Jennings, "Agent Theories, Architectures, and Languages: A Survey", in Proc. European Conference on Artif. Intell., Workshop on Agent Theories, Architectures, and Languages, M. Wooldridge and N.R. Jennings eds, Springer-Verlag, LNAI, vol. 890, Heidelberg, Germany, 1994, pp. 1-39.

Modeling MultiAgent Systems as Self-Organized Critical Systems

Pierre Marcenac

[Marcenac 98b]

Accepted Paper

31st Hawaii International Conference on System Sciences, HICSS-31, IEEE Computer Society Press, Jan. 1998.

Modeling MultiAgent Systems as Self-Organized Critical Systems

Pierre Marcenac
MAS2 Team - IREMIA, University of La Réunion, BP 7151
97715 St Denis Cedex 9 - France
E-mail: marcenac@univ-reunion.fr

Abstract

The general framework of our project is to provide a computational model of physical complex processes for simulation needs. In Geophysics, the study of this kind of system has led to the concept of "Self-Organized Criticality", to explain the "repeatability" of emergent phenomena in nature. To model Self-organized criticality within computers, the original part of the work is to propose a multiagent platform where emergent phenomena are dynamically created during simulation at the time they occur. The aim of this paper is to show that multiagent systems, studied as "emergent systems", can help in providing adequate mechanisms needed to model self-organization in complex systems. The paper introduces some key issues associated with the understanding of intrinsic mechanisms leading to self-organization and discusses of the implementation of such mechanisms in an agent-architecture. Finally, it describes a life-sized experimentation in earthquakes simulation to validate it.

1. Introduction

This paper reports an interesting work integrated in a larger project (MultiAgent Systems, Modeling And Simulation - MAS2) which aims at providing a computational model of physical complex processes for simulation needs. This kind of tool will be meant for researchers needing to simulate complex physical models, without having to implement them. The purpose is to provide a complete toolkit as a virtual laboratory in order to design a large scope of dynamic systems, as well as interfaces to set and control the simulation. The processes which have been already studied in the project are those of natural phenomena in Geophysics, such as earthquakes and volcano eruptions.

In natural phenomena modeling, simulation is a very interesting feature to investigate, because it can adequately capture any behavior likely to be observed, and is used to exhibit remarkable parameters or structures and tackle their role in the system. To try to understand such complex phenomena, the classical approach is to aggregate data, knowledge and hypotheses to build a model of the physical world. This model is generally expressed by mathematical relationships between variables, matching some real physical magnitudes, such as differential equations for instance. Ran on computers, simulations provide tests to validate the theoretical model. Results can then be processed and exploited with the help of statistical techniques to verify the given hypotheses.

In order to explore complex physical processes models, the framework of our project is to model and simulate the behavior of complex systems with an agent-based approach. Complexity is a property found in many kinds of natural systems, in physics, biology, social sciences... It is at the crossroads of different approaches, system theory, artificial life, cybernetics, artificial intelligence...

According to the oxford dictionary, the complexity of a system involves two or more components which are (1) joined in such a way that is difficult to separate them, and (2) closely connected [3]. This duality determines two dimensions of complexity which lead to the same approach in computational modeling: the distinction in several components leads to study the components structure, as the connection leads to study the interactions dynamic. A complex system is inwardly driven by interactions between its components whose result exceed the contributions of individual components.

In such a frame, the agent paradigm seems to be an interesting approach to investigate, because it provides a computational model which naturally aims at representing local phenomena, and in which the emergence of the system's behavior is the result of interactions between agents.

In accordance with J. Ferber's definition, "an agent is a physical or potential entity, acting on itself and its environment and disposing of a partial representation of this environment. An agent pursues an individual goal, communicates with other agents, its behavior being the

consequence of its competence and communications with others" [9]. As the agent pursues an individual goal, it is assumed to be *independent* and can process tasks without explicitly receiving the order. Actions are performed through asynchronous messages passing, and an agent does not wait for another resource to proceed. At the same time, an agent is given an *autonomous* part which can control its behavior: the autonomous part determines the agent's self-governing: an agent is able to take an initiative and exercise a flexible degree of control through its own actions, by dynamically choosing which actions to invoke.A complex system, assumed to be non predictable and non-linear, is then cut in tiny pieces in an adequate complexity level. The agent paradigm, representing the real world as autonomous parts, is very encouraging in this field. This consideration is based on a previous project led in our team for three years, dealing with volcano eruptions modeling, in which quantified results were produced from local interactions between agents, each one seen as a physical component [15], [22].

This article rather focuses on the inverse relationship, that is how multiagent systems, studied as "emergent systems" can help in providing adequate mechanisms needed for complex systems modeling and simulation. Emergence is here understood as self-organization relative, that is the appearance, in a specific context and in an active environment, of new phenomena not previously identified, and from now on irreversible within a system of interacting entities.

In geophysical complex systems, this point of view has led to the concept of *Self-Organized Criticality* (SOC) [1], to explain the "repeatability" of phenomena in nature, which can obviously be observed in scaling laws. Such systems are driven by highly non-linear behavior, where a small external perturbation could generate a large-scale phenomenon at a critical state of the system, but without predicting when it could appear. One of the important results of this approach is to consider the critical state as an "attractor" for the system's dynamics. This gives importance to individual actions which work towards the elaboration of the phenomenon, and therefore its organization.

One of the best well-known example illustrated throughout the paper is the avalanches frequency in a sand heap, where grains are randomly added during experimentation [1], [3]. Such a system describes a degree of complexity more important that its parts, and includes properties which could not be reduced to those of its components. This non reduction is assigned to the presence of interactions which dynamically unify the components of the system at the critical state, and from which an avalanche appears by affecting a part of the system.

The original part of this work is to propose a multiagent platform where emergent phenomena are dynamically created during simulation when appearing. The simulation result is then assimilated, on the one hand, to quantified results, and on the other hand, to the appearance of new patterns which model phenomena. In this paper, we investigate how natural phenomena can be represented as emergent structures created by self-organization within a multiagent system.

Dynamically creating structures is an interesting feature, because it allows the system with keeping trace of phenomena and being adapted throughout the simulation. For instance in natural phenomena (earthquakes or volcano eruptions for instances), the affected part of the system can in its turn force the components, and plays a role in the system for future behavior (for instance, there is no eruption, nor earthquakes twice a time at the same place). From a computer science point of view, this approach seems very useful to understand (1) mechanisms leading to self-organization, and (2) how a complex system can be modeled to take emergent phenomena into account for future events. Our assumption is then based on the fact that once appears, an emergent phenomenon becomes intrinsic to the system, and its new characteristics can no longer be inferred as before.

To achieve such a goal, the article points out the definition of emergent phenomena as being viewer relative by characterizing different kinds of "observers", and describes how self-organized critical systems can be modeled by successive abstraction layers in an agent architecture, each higher one encapsulating the emergent behavior of the micro-level below it.

The paper is divided in three main sections. Section 2 introduces some key issues associated with the understanding of intrinsic mechanisms leading to self-organization, by trying to answer the question of how an artificial system, composed of simple agents interacting each others, can generate emergent structures. Section 3 discusses the implementation of such self-organization mechanisms in an agent-architecture and argues the introduction of an intermediate abstraction level needed to model emergent phenomena resulting of agents' interactions. Finally, section 4 describes a life-sized experimentation in earthquakes simulation to validate both the architecture and the associated mechanisms.

2. Self-organization mechanisms

Self-organization allows a system to be organized or re-organized during time, as far as the simulation processes. Self-organization is a kind of emergence, because the pattern which is organized is (1) dynamically generated from local interactions, and (2) determines the behavior of underlying agents which have given birth to it. This second property defines the system adaptation: adaptation is then viewed as a consequence of self-organization: it refers to agents' behavior which must be adapted because the environment is changed or moved. Adaptation is then defined as a capability to improve and adjust behavior to the environment.

In complex systems modeling, self-organization is a very interesting feature, and has been tackled in several domains. For instance in Socio-Biology, where the collective organization of social insects is shown highly

sophisticated, as individual organization is limited and apparently random [30]. In Geology and Geophysics, self-organization is intrinsic to self-organized critical systems. In Sociology, some works aim at understanding emergence of organized societies, and explain transitions from small families to large groups and tribes or relationships between institutions and individual phenomena such as beliefs [13].

The following sections introduce the basis of emergence and self-organization in multiagent systems, that is a set of mechanisms able to analyze the conditions of a phenomenon appearance, and to observe it.

Section 2.1 introduces some fundamentals of emergence in artificial systems through causal links and complexity grain levels. Section 2.2 details how a phenomenon can emerge in a multiagent context and describes the conditions which will trigger the emergence and the principles needed to provide the general framework of self-organization. Section 2.3 discusses the necessity of an "observer", intended to answer the question of how can we observe that self-organization took place. This observer should make emergent phenomena intrinsic in the system, and study the conditions under which the phenomenon stops.

2.1. Emergence := causal links + scale change

There is no common ontology in emergence, and it has been differently tackled in related works, according to the area it has been studied:

The most famous are C. Langton's works on artificial life [19], focusing on the artificial system[10] structure, defined by its effects in a given environment while functional aspects are often neglected.

S. Forrest's works [11] on emergent computation are also popular; emergent computation only uses local information to generate emergent behavior. Emergent computation is at the same level of difference with standard computation as linear systems with nonlinear systems in Physics. It focuses more on dynamic behavior rather than static data structures. Some models of emergent computations have been implemented, for instance, in CCM (Chemical Casting Model) [16], a simulated reaction system looking like a chemical science experience, which is used to process computational properties. Such a system is solving classical problems of computer science, such as sorting, parity checking or prime number computation for instances.

In Psychology, J.R. Searle's works on consciousness are interesting too [26]. J. Searle argues that emergence creates new functions where there were not, for instance, consciousness allows flexibility, sensibility and creativity. His results describe two levels of emergence, according to their "explicability" level. Other related areas have been studied, for instance in social science [8].

The unified concepts of these works is that the appearance of structures in a component C is seen as the consequence of dynamic interactions, called *causal links*, between fine-grain components of C, and involving a scale change:

Causal links are dynamic conceptual links which define interactions (interactions are understood hereafter as a dynamic relationship through mutual actions) between components in order to satisfy a common goal.

The scale change introduces the representation of grain levels in complexity: low level describes fine-grain components of n-complexity, as high level describes coarse-grain components of $n+1$-complexity. An emergent phenomenon is caused by fine-grain components and is observed in higher levels.

In complex systems, fine-grain components are atomic, that is to say indivisible. In this level, the entities behavior is determinist and is described by elementary rules which consist in reacting to environment disturbances (stimuli/reactions). Each component does not dispose of symbolic representation of the whole world in which they evolve. Coarse-grain components are not determinist, because their behavior is not linear during time. Such a decomposition is familiar in multiagent systems, where fine-grain components are associated to *reactive* agents embedded in a higher abstraction level, the *agents' society* [9].

By translating causal link and scale change notions in multiagent systems, we can model emergence as the result of an organized activity of interactions between (1) coarse-grain agent and its components, (2) fine-grain agents, and (3) fine-grain agents and its society. Figure 1. illustrates such a purpose:

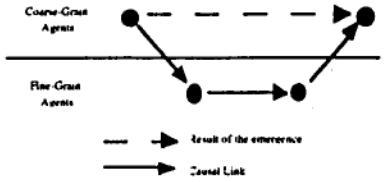

Figure 1. An emergence situation

So, if only one causal link exists between n+1-complexity and n-complexity levels, there is no emergence (macro-behavior can be reduced to micro-behaviors). Emergence is then understood as successive "causes" of n+1-complexity to n-complexity, then n-complexity to n-complexity, and finally n-complexity to n+1-complexity. It is based on local interactions between agents and therefore involves dynamic aspects of the system. This dynamic action assumes that somehow, a causal result in a coarse-grain agent is more than the sum of its components' local interactions [27]. Moreover, once a phenomenon has emerged, it exists without any reference to underlying agents which have given birth to it. This property involves that emergent phenomena become intrinsic into pre-existing systems, and therefore that new characteristics can no longer be inferred from fine-grain components characteristics [17].

[10] Artificial systems are defined as systems achieving emergence to model, simulate and finally generate more and more complex behaviors [2].

2.2. Intentional emergence: artificial emergence

To translate self-organization concepts in a computational model, we need to define:

Mechanisms in charge of detecting the conditions which will trigger the phenomenon and stop the process when finished. These mechanisms act at a local level and are then defined as a part of the agent's behavior.

A mechanism to give birth to the phenomenon, and in charge of its representation and visualization though the system. This mechanism looks like a "*meta-function*", which must evaluate pre-conditions given by the trigger, and manage the emergent structure. It acts at a global level and is then defined as a part of the agents' society.

However, we claim that this kind of emergence should be understood as "simulated", because the way how such structures occur is totally described by a determinist process (as results of such processes are not). In the area of complex systems, we argue that, because researchers do not actually have sufficient knowledge, it is not yet possible to formally explain the results. One of the consequences is that emergence can be perceived as "*built*", and we believe that a real and strong emergence (so-called "*high emergence*" in the following by opposition to "*built emergence*") is not an intrinsic characteristic of artificial systems. Emergent phenomena in artificial systems could be explained, but it is rather because actual human-knowledge is not sufficiently advanced to do so. High emergence is considered as a characteristic of natural systems rather than artificial systems, a natural system defining an unclosed world, in which all situations are not known by the designer. In such systems, natural phenomena are highly emergent due to ignorance of governing laws.

We then agree with Mitchell and Hofstadter's dissertation in [25], who specify that explanations about emergence can only be done at a micro-level, because human-knowledge fails when trying to explain it at a macro-level. This is why a phenomenon, emerging in an agent, can not be explained from the study of its underlying fine-grain components. This consideration has been well-studied in general system theory by Ludwig von Bertalanffy [28].

We refer hereafter as "*artificial emergence*" the emergence property in artificial systems, provided by a meta-function from the local analysis of both environment and agents' current states. Finding architectures to make use of such mechanisms is a very interesting feature which can help in better understanding artificial emergence.

2.3. The role of the observer

An observer is a mechanism in charge of the observation of the self-organized phenomenon. This observation is essential in simulation, as it constitutes a natural interpretation of the results of the emergence. The observation must be *visible* from an external point of view or *programmed*. These considerations distinguish two kind of observers:

An external human observer, for instance a end-user looking at the simulation results; for him, the results could be often surprising and seen as a kind of "magic phenomenon", as understanding the reasons for which structures emerge is indescribable and impossible to formalize [10]. This constitutes the external point of view of the emergent phenomenon, where the system surprises the user who only knows what occurs at the finest grain level. R. Brooks considers in that case, that emergence exists only when it is necessary to use new words to describe what is observed, and where these terms are not described in the fine grain level of the system [5]. This point of view does not impose to store emergent phenomenon within the system, as operating on the emergence results is not required.

At the opposite, when the observation takes place inside the system, saving emergent results is obviously needed. This is moreover one of the most interest for considering the observer as programmed. The designer knows what external observers are looking for, but does not know neither the exact result of the simulation, nor what will be exactly emerged from the simulation. So, for him, there is no magic phenomenon, as he controls the emergence capability of the artificial system. In addition, he describes internal actions of each agent to do so. For S. Forrest and J.H. Miller [12], if the result of interactions between agents leads to a non predictable effect for the designer himself, the emergence must be qualified as high. The observer programmed is integrated as a part of the system and is distributed among all agents. This observer gives the internal point of view of the system, this is the reason why there is no magic part in artificial emergence. Its role is to put in mechanisms detecting the end of emergence process.

The next section presents the agent architecture implementing artificial emergence with the help of a detection mechanism and a programmed observer both acting at a local level, and a meta-function acting at the global one.

3. Modeling SOC systems with agents

3.1. Modeling catastrophes as medium-agents

Traditional multiagent architecture intended to model complex systems describe two levels:

The first level corresponds to a coarse-grain agent, and describes the macro-level, as a related element in which global solutions will be observed. It holds on the underlying agents' organization in a society. In multiagent systems, such an organization has no reason to be if it is unable to represent interactions facilities between agents. The society is therefore organized as a network of acquaintances, managing agents in the system to match global specifications. The role of the whole system is expressed through an external behavior, and driven by an input/output interface with the external world (software, human...).

The second level corresponds to fine-grain agents. It describes the micro-level as reactive agents. Each reactive agent is described by interactions, micro-behaviors and evolution capabilities. At run time, agents will interact in a concurrent way with their surroundings. The micro-level masks the complexity of the whole system which is encapsulated in agents describing low-level and basic actions. Such reactive agents are often called *cells* in some related areas (Biology or Medicine for instances) or *micro-agents*, in a more general way. A micro-agent is reactive, as it generates behavior without explicit representation of the domain neither global task constraints. micro-agents just interact with their surroundings, and evolve during time by updating their internal state. They get close to each other, and form a compact group. They act in response to events that are too fine-grained to be understood (or explained) by a macro-agent. Interactions between micro-agents are thus provided by signals which do not bear semantics, their meaning depending on the interpretative ability of the receiver.

It is then tempting to compare a complex system with such a multiagent system, and to associate real world components with micro-agents, and the whole system with the multiagent system. For instance in the sand heap example, sand grains will be modeled as micro-agents, as the heap will constitute the agents' society.

However, as we seen it before, a self-organized critical system should integrate the non predictable dimension: the critical state of the system leads to catastrophes (for instance, a only one grain can generate an avalanche in a sand heap). Our assumptions are then to consider the critical state as assigned to particular states of micro-agents, and the avalanche to *a spatio-temporal aggregation*, in some particular context and environment, and under specific conditions of micro-agents taking part in the phenomenon. The result of this aggregation is then an emergent structure created by self-organization in the multiagent system. This approach seems handy, because it allows to consider the catastrophe as an entire entity able to compute its own properties. Moreover, it is at the right place to distribute new computed values after the critical point in order to constraint the affected part of the system. Thus, in this approach a sand avalanche will be modeled by a structure composed of sand grains (micro-agents) which were in a critical state just before sliding.

Therefore, the main issue is to look for computational model allowing to represent and characterize self-organization, from local interactions between agents. Considering a traditional two-levels architecture in the system makes the catastrophes modeling task difficult in the system because:

On the one hand, as a micro-level is described by fine-grain agents, the organization of new structures can not be processed in such a microscopic level, which does not have enough knowledge to do so. Beyond this microscopic dimension, emergent structures can only arise in one macroscopic level, disposing of missing knowledge.

On the other hand, it is often impossible for a macro-level to characterize and correctly identify the whole set of emergent structures. As the complex system is non predictable and badly understood, trying to explain emergence mechanisms at the global level of the system fails. As a matter of fact, a significant number of micro-agents could intervene in the whole emergence process, making the system too complex to be analyzed at a macro-level.

It is then skillful to introduce an intermediate grain level between the macro-level and the micro-level, embedded as sub-organizations. Such an intermediate level is called *medium level* and sub-organizations, *medium-agents*. The medium level is designed before the simulation begins by giving sub-organizations' structure and behavior. This point of view enforces the necessity of finding the most efficient abstraction when designing this level, which is totally dependent of the context tackled [6]. However, medium-agents are not dynamically pre-defined in the system because they spontaneously appear during the simulation as the result of self-organization.

When self-organization arises (meaning that the system's critical state is hit), a new medium-agent will be created within the system to model the catastrophe. This new agent describes a society of micro-agents which have given birth to it, and at the same time, is viewed as an agent of the society defined in the macro-level. This duality expresses a recursive view of an agent in the architecture. Medium-agent's properties are computed and set when the organization has been built. Such properties are then "re-introduced" in underlying lower-level agents as constraints to apply in their own structure, this mechanism being called "*back-propagation*" (see next paragraph). By this way, a medium-agent is playing its causal role for the rest of the simulation, by constraining a part of the system. Figure 2. illustrates this intermediate level in the architecture:

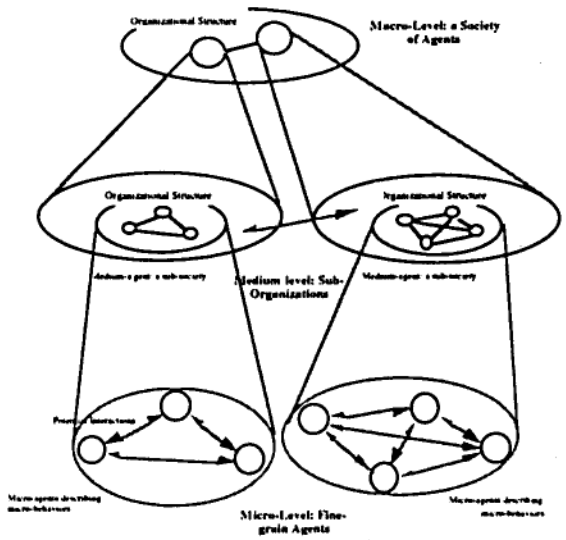

Figure 2. Medium-agents

The three next sections present the implementation of self-organization mechanisms in the architecture. Section

3.2 describes the trigger mechanism which locally detects necessary conditions to initialize the whole process, section 3.3 discusses of the meta-function used to generate the new structure, and section 3.4 proposes the programmed observer as a end-detection mechanism. Section 3.5 summarizes how these mechanisms are working on the sand heap example which illustrates concepts throughout the paragraph. Finally, section 3.6 briefly compares this approach with those of related works.

3.2. The trigger mechanism

In our architecture, local interactions between micro-agents, at the origin of self-organization, result in a modification of internal properties which define the agent's state. The agent's state is represented with a "state vector" P, from which each coordinate describes an internal property:

$$P = \begin{bmatrix} P1 \\ P2 \\ . \\ . \\ Pn \end{bmatrix}$$ where p_i is an internal property of the agent.

A subset of these internal properties, called *state parameters*, are limited by *thresholds* giving the agent's critical state. Thresholds and state parameters are set during the design phase of the application. An agent is then stable when state parameters values do not reach the thresholds.

In the architecture, the trigger mechanism is distributed over micro-agents in order to detect *similarity* over state vectors through neighbor agents being in a critical state. During the simulation, if a threshold is hit, the constraint is transferred to neighboring micro-agents, sometimes generating avalanche behavior, and at the same time, similarities are searched. Similarity is based on the comparison of each agent's state with those of its neighbors in order to detect common states. If an agent $Ag1$ is recognized as similar with a neighbor's one $Ag2$, we consider that both agents have organized themselves for a common purpose. The two agents will then be grouped together and embedded as a self-organized structure by the meta-function.

$Ag2$ looks for similarity in its turn, by asking neighborhood. The detection mechanism is therefore propagated to all agents in the system. The result of this research provides the set of agents whose critical state is exceeded, that is all agents taking part to an emergent phenomenon.

To determine if two agents A_p and A_q are similar, it is only necessary and sufficient that:

A_p and A_q are neighbors,

A_p's and A_q's state vectors are respectively:

$$P = \begin{bmatrix} P1 \\ P2 \\ . \\ . \\ Pn \end{bmatrix} \text{ and } Q = \begin{bmatrix} Q1 \\ Q2 \\ \\ \\ Qn \end{bmatrix}$$

where $\forall i \in \{1...n\}, |p_i - q_i| < \varepsilon_i$

ε_i defines a set of accuracy factors identifying the degree of similarity between properties p_i and q_i. This parameter is globally defined before the simulation begins, allowing the user with customizing the application for specific requirements.

Thus, similarity provides a way to detect agents which contribute to emergent phenomena, when a constraint is distributed within the system. In addition, it can be interpreted with a margin of error ε_i which can be chosen according to the context tackled. In our system, similarity provides good results because the geophysical context does not need to represent a complex cooperation between agents, as in [7] for instance, where social relationships between two agents could determine an organizational commitment. In such artificial societies, the use of other approaches, such as complementary or antagonism could be more adequate [13].

Two micro-agents which are mutually self-identified as similar are forming a class of similarity by self-organization. A class of similarity then characterizes some organizational structure linking agents with similar ones. A global mechanism is then needed because, when a similarity is detected, the trigger mechanism does not have enough knowledge on the global structure and can not then "aggregate" agents at the right place. Such a mechanism is described by the meta-function detailed in section 3.3.

3.3. The meta-function

The meta-function takes place each time the trigger mechanism detects a similarity between two or more agents. The meta-function must be global and is defined as a part of the society behavior. This global mechanism is in charge of creating a medium-agent to aggregate similar micro-agents, or to add such agents to an existing one, as similarities are detected.

If one of the two similar agents is already member of a sub-organization, the meta-function advises the underlying medium-agent managing the sub-organization, which integrates the new micro-agent within the existing structure. Note that if the two micro-agents are members of two different sub-organizations, they can be integrated on both sub-organizations, to take different emergent phenomena into account. If no medium-agent has already been created to model the class of similarity, the meta-function is creating a new medium-agent responsible of the emergent structure. This agent will be then populated step by step. Finally, note that such an encapsulated micro-agent can leave the structure at any time, if its state evolves again. If only one agent remains in the structure, the sub-organization is deleted by the meta-function.

Beyond this mechanism, a medium-agent can react as a feedback on each member of its structure, to introduce constraints which concern all agents in the organization. This feature is called a *back-propagation*. At the design phase of the application, the designer gives the void structure of a medium-agent defining its properties, which are independent of micro-agents' ones. During the simulation, when the medium-agent is entirely populated,

the meta-function computes appropriate values for the new agent's state vector. Setting new properties in a medium-agent becomes easy at this point, mainly because the meta-function picks up information given by the trigger mechanism from lower-level. This set of new properties defines a kind of constraint or restriction governing the new structure. This constraint acts as a global law for all underlying micro-agents belonging to the structure, and back-propagation is the way to inform such micro-agents.

Therefore, when the emergent structure back-propagates a constraint, it forces the behavior of underlying agents by applying them some of its properties. This view is very close to reality: during a sand avalanche for instance, each grain releases energy and moves into the system. the remaining energy in sand grains is very low, and micro-agents modeling sand grains should be properly re-initialized so that the whole system behavior might be modified accordingly.

3.4. The programmed observer

Finally, the role of the programmed observer is to look at the end of an emergent phenomenon when remaining micro-agents' internal properties are too low to hit their thresholds. The system becomes stable again, and this situation can be easily observed when similarity detection fails, that is the reason why it is interesting to embed it in micro-agents. The programmed observer acts as a end-detection mechanism, by examining the neighborhood to look at potential agents not enough stable (and able to propagate again the phenomenon). The end-detection mechanism consists in computing the number of micro-agents remaining unstable. Thus, when the counter becomes nil, the end of propagation is detected.

In traditional multiagent systems, the programmed observer is generally implemented as a global loop inserted over the multiagent system or as a part of the society, to inspect the stability of underlying agents forming the structure. In our architecture, the programmed observer is distributed and locally defined, to remain true to the agent approach.

3.5. How does it work on a simple example?

If we look at the sand heap example, each grain can be modeled by a micro-agent with a state vector describing internal properties of the grain, such as density and force limited by a resistance threshold. When a sand grain falls down, it provides a new constraint to be applied to neighbor micro-agents. The constraint is propagated throughout the network by local interactions between micro-agents. The constraint is then locally performed and each agent computes the new force. This point of view is well appropriate to natural phenomena simulation: external events, simulating constraints applied on the structure, are performed by each micro-agent. Each micro-agent reacts by changing state and calculating the remaining constraint to spread over the network. As the constraint intensity is decreasing when micro-agents perform the constraint, a catastrophe is not always caused by the external event, that is the reason why the system can never be determinist.

However, if an agent's force becomes greater than its resistance threshold, the trigger mechanism looks at the neighborhood for others agents which are in a similar state. As soon as two agents are recognized similar by the trigger mechanism, an avalanche is going off. The meta-function then takes over and creates the medium-agent to aggregate all agents being involved in the phenomenon. The observer asynchronously tries to locate the end of the avalanche by looking at agents' states to check stability, (if force remains high or falls down the resistance).

When the end-detection mechanism is ensured the end of the phenomenon, the medium-agent can then compute some properties linked with the avalanche: total amount of energy released or avalanche areas for instances. It can also compute some parameters which directly affect sand grains, remaining energy for instance. These parameters are then applied to micro-agents by the back-propagation mechanism, to re-initialize sand grains properties and position.

With this kind of approach, the system is able to easily provide some results dealing with avalanches, and to "memorize" the phenomenon; it is now ready for future disturbances and is closely related to the reality.

3.6. Comparison with other approaches

We have enumerated three grain levels to describe local interactions and identify global emergent processes. Thus, we have pointed out that both a micro-level and a macro-level was not sufficient to observe and control emergent phenomena. We have therefore introduced an intermediate level including medium-agents to achieve such a function. We have seen that medium-agents are spontaneously created by a schedule of bottom-up, before being homogenized by top-down mechanisms.

The whole self-organization process of our model acts both at a local and global level. The method used to initialize the emergence is based on the similarity detection. The macro-agent is then immediately warned about self-organization and generates a medium-agent modeling the emergent phenomenon. This approach is interesting when tackling physical processe because it allows:

- to model the behavior of complex systems assimilated to self-organized critical systems during simulation,
- to represent emergent phenomena as new structures,
- and to take account new constraints which can appear as the result of the emergence process.

In the MANTA system [8] which simulates the socio-genesis of an ants colony, the different entities of the colony's life are modeled by micro-agents at a micro-level. The adaptive performance of the society results in the basic behavior of its members. In the MANTA system, the generation of social patterns in the ants' society can be observed, but any explicit structure is defined to organize

them. This is related to the fact that no further information is needed when the phenomenon appears, as the observer is external to the system. In our system, information is needed for back-propagation, and the observer is then belonging to the system (that is the reason why we called it a "programmed" observer).

In the SWARM system [24], the agent's behavior is defined by the emergent phenomena of the agents inside its swarm. However, a swarm does not explicitly dispose of any organizational structure for the same reasons as in MANTA.

Lastly, cellular automata constitute an alternative approach of parallel computation, which allows to create virtual worlds represented by matrix, evolving with the help of sample local rules. Emergent structures are then observed when forming topologic structures within the matrix. However, the model is too simple to implement "emergent systems", because the system behavior can not be driven by properties which derive from the self-organization process.

The next section presents an example illustrating such mechanisms in the context of a life-sized experimentation.

4. A life-sized experimentation

4.1. Modeling earthquakes with SOC

This section presents a short validation of the previous mechanisms with an application built in order to simulate complex mechanics of earthquakes, such as those observed in San-Andrea, California. The application proposes to model a geophysical system representing a "break", composed of rock items, and subject to earth's crust constraints. Each rock item is characterized by rheologic properties which constitute internal data, represented in the state vector. Micro-behavior is defined to react to disturbances by fitting the state vector. When a micro-agent becomes unstable due to disturbance, it releases some energy and, if the amount of energy is sufficient, propagates the disturbance to the neighborhood.

This propagation can hit the system critical point, leading to earthquakes in the break, which will be represented by emergent structures. This experimentation has shown how a non-initially constrained system can be organized as events arrive. This is specially useful in the frame of natural phenomena such in earthquakes prediction, as researchers are attentive at spontaneous emergence of order in a system which looks like randomly initialized [18].

At the beginning of the simulation, the system is represented by a macro-agent and micro-agents modeling the break and rocks. Micro-agents representing rocks are defined with a state vector randomly initialized, and the stability of internal properties is given by some thresholds:

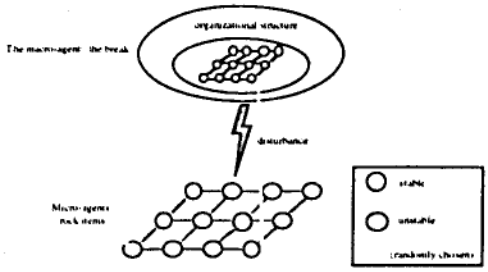

Figure 3. Break in the multiagent system

The disturbance is initially distributed among rocks items and is asynchronously interpreted by each micro-agent which reacts by setting its state vector and releases energy to each neighbor, causing a disturbance in series. When interaction occurs, the local trigger mechanism is then set each time a threshold is hit to detect similarities between unstable micro-agents. Such similar agents will progressively generate the earthquake, and will be aggregated as a medium-agent by the meta-function:

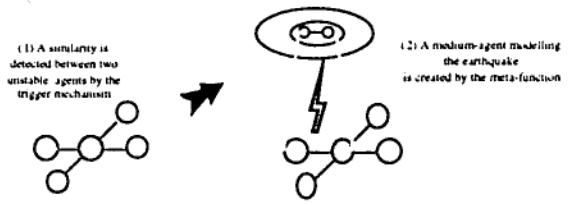

Figure 4. Similarity-detection

The end-detection mechanism (the programmed observer), embedded into micro-agents, examines the potential number of rock items which remain unstable and can still propagate the phenomenon (marked with a black point in Figure 5 below):

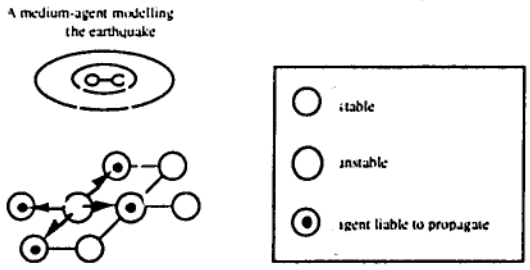

Figure 5. End detection

Finally, after all micro-agents are stabilized, no more agent is enough constrained to take part in the earthquake, and the structure is completely defined and integrated in the system, as shown in Figure 6:

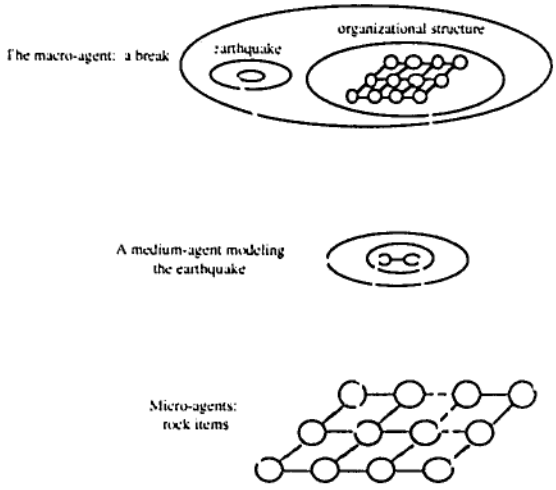

Figure 6. Giving birth to earthquakes

Note that the new structure is now bearing new properties, such as the earthquake size and magnitude for instance, which are computed by the meta-function when the earthquake stops. This is important to take into account, as such a disturbed system can not behave as before earthquakes appear. So, these properties will then be distributed again on each agent making up the earthquake, as far as the simulation processes.

In addition, the recursion property of multiagent systems is here preserved: the emerging structure of the earthquake is now on irreversible, and is considered both as an agent of the society in the macro-level, and as a sub-society of agents in the medium level, composed of old unstable micro-agents which have given birth to it.

4.2. Implementation issues and brief results

This application has been implemented with the help of ReActalk [14], an open environment to experiment with agents. ReActalk is a platform for reflective actors in Smalltalk, and is an extension of Actalk [4], a platform for the study of actor paradigms within Smalltalk-80.

Smalltalk-80 provides the structural reflection as well as the minimal object environment, and Actalk provides their asynchronous manipulation. However, an actor is different from an agent because an actor does not take into account its system membership when an agent is necessarily indissociable of its environment. In addition, an actor neglects the autonomous part of the agent: it does not control its behavior nor allow to choose the best way to achieve a given task, because it is too linked with the class which has defined it. From these establishments, S. Giroux has developed a reflective approach of an actor to implement the notion of agent. Therefore, an agent is seen as a reflective actor, where reflection is now "operative", allowing an actor to be cut off from the class which gave birth to it. Thus, this approach gives an actor autonomous and evolution capabilities. From our point of view, a reflective actor models: (1) the components of a system as objects, (2) their concurrent evolution as actors and (3) their autonomy and independence with operative reflection.

A graphical interface has been realized in Smalltalk-80 to set parameters and to follow the simulation as earthquakes appear. Our agent-based approach has been validated on the Miller's model [23] of a break, subjected to earth's crust constraints. From local basic behaviors, the model implemented has reproduced the break complexity at a global scale. Recovering in-situ observations from synthetic data helps in validating both the architecture and the associated self-organization processes. Further experimentation has been led to simulate a model of a fluid's tank (in a porous environment), encapsulated in a matrix of rocks and the same kind of results have been reported.

5. Concluding remarks

This paper has presented a model of self-organization in artificial systems intended to model self-organized critical systems. Our approach is to consider the program execution as a set of interactions among agents. Knowledge divisions which help to reduce a design's intricacy when tackling complex systems is first reminded, and a specific architecture allowing to distribute the complexity in independent and autonomous agents is provided with that aim.

In this work, we have more particularly focused on the study of multiagent systems seen as emergent systems, which is an interesting approach to model complex systems based on self-organized criticality. In such an approach, emergent phenomena are dynamically represented within the system. To achieve such a dynamic self-organization process, we have derived appropriate computational mechanisms on the architecture, such as a trigger, a meta-function, and an observer.

This approach was proved to be very helpful in understanding and analyzing the behavior of earthquakes in a life-sized context. The global system behavior has been reproduced from a model of very simple micro-agents' behavior. This global behavior was unattainable up to there with more classical approaches. The architecture then validates the self-organization approach for complex system modeling and simulation. We are now convinced that the architecture is appropriate for other natural systems simulation exhibiting similar behavior, and we actually experiment the approach in the social behavior framework.

Furthermore, we aim to extend our experimentation to the field of artificial life, and complex adaptive systems. With that aim, further researches are now investigated:

First, though systems validated in this context are quite small, during simulations, agents are faced much of time to the same kind of conditions. The repetition of the same actions leads the system to become less powerful. To address this issue, the system has to analyze the conditions which drive the best actions. Primary tests were first introduced in our architecture to look for the convergence of certainty factors when agents broadcast messages everywhere in the agents network [20]. It is then necessary

that multiagent systems should be equipped with the ability to automatically improve their future performance.

Second, we are investigating adaptation mechanisms with machine-learning techniques, such as classifiers and genetic programming. Machine-learning techniques are good candidates for making the system more intelligent and better drive the emergence process. This should allow to model intelligent artificial entities, and tackle new fields as biologic and sociologic systems. This remark is complementary to the first one, as this approach should significantly improve the performance of the whole system: thanks to the observation of similar situations, such techniques could generate rules allowing an agent with choosing the most adapted behavior to perform. By derivation, if such rules have been learned by a macro-agent, some inputs events can also be intercepted and locally performed by this mechanism to avoid low-level computation.

However, learning in multiagent systems is more difficult than in traditional artificial intelligence systems, mainly because we need to take into account collective behavior. Learning must be distributed among agents, and implemented with the same mechanism as the agent itself [29]. The architecture has been re-engineering and re-implemented in JAVA in 1997 with the aim of keeping researchers close to such goals.

Acknowledgments

We are grateful to those people who have brought an important contribution to this work, and more particularly S. Calderoni and S. Giroux.

References

[1] P. Bak, C. Tang, K. Wiesenfeld, "Self-Organized criticality: An explanation of 1/f Noise", *Physical review Letter*, 59(4):381-384, 1987.

[2] E. Bonabeau, J.L. Dessalles, A. Grumbach, "Characterizing emergent phenomena (1) and (2).: A critical review", in *Revue internationale de systémique*, 9(3):327-346 and 347-371, 1995.

[3] E. Bonabeau, P. Bourgine, "Artificial life as it could be", *World Futures*, (40):227-249, 1994.

[4] J.P. Briot, Actalk: "a testbed for classifying and designing actor languages in the smalltalk-80 environment", in *Proceedings of European Conference on Object-Oriented Programming (ECOOP)*, Cambridge, England, 1989.

[5] R.A. Brooks, "A Robot that Walks: Emergent Behaviors From a Carefully Evolved Network", *A.I Memo* (1091), Massachusetts Institute of Technology, Cambridge, MA, USA, 1989.

[6] S. Calderoni, P. Marcenac, "Emergence of Earthquakes by MultiAgent Simulation", *European Simulation Multi-Conference*, ESM'97, Turquey, pp. 665-669, 1997.

[7] C. Castelfranchi, "Commitments: from individual intentions to groups and organizations", in Proceedings of the *International Conference on MultiAgent Systems*, ICMAS'95, December 1995.

[8] A. Drogoul, J. Ferber, "Multi-Agent Simulation as a tool for Modelling Societies: Application to Social Differentiation in Ant colonies", *Artificial Social Systems*, (830), Springer-Verlag, C. Castelfranchi and E. Werner eds, pp. 3-23, 1994.

[9] J. Ferber, "Reactive Distributed Artificial Intelligence: Principles and Applications", in *Foundations of Distributed Artificial Intelligence*, N. Jennings eds, North-Holland, 1994.

[10] J. Ferber, "Cooperation Strategies in Collective Intelligence", in Proceedings of the *Workshop on Modelling Autonomous Agents in a Multi-Agent World*, MAAMAW'96, The Netherlands, January 1996.

[11] S. Forrest, "emergent computation: Self-organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks", introduction to the proceedings of the *ninth annual CNLS Conference*, in emergent computation, MIT Press, Cambridge, MA, pp 1-11, 1990.

[12] S. Forrest, J.H. Miller, "Emergent behavior in classifier systems", *Emergent computation*, MIT Press, Cambridge, MA, USA, pp. 213-227, 1990.

[13] N. Gilbert, J. Doran, *Simulating Societies: the computer simulation of social processes*, N. Gilbert et J. Doran eds, UCL Press, 1994.

[14] S. Giroux, "Open Reflective Agents", in Intelligent Agents (II), *Agent Theories, Architectures, and Languages*, M. Wooldridge, J.P. Müller and M. Tambe eds, Springer-Verlag, LNAI, (1037), Montreal, Canada, pp. 315-330, 1996.

[15] S. Giroux, P. Marcenac, S. Calderoni, D. Grosser, J.R. Grasso, "A report of a Case Study with Agents in Simulation", in proceedings of the *International Conference on Pratical Applications of Intelligent Agents and MultiAgent Technology*, PAAM'96, London, UK, pp. 295-310, April 1996.

[16] Y. Kanada, M. Hirokawa, "Stochastic Problem Solving by Local Computation based on Self-organization Paradigm", in proceedings of the *Hawaii International Conference on System Sciences*, pp. 82-91, 1994.

[17] G. Kampis, *Self-modifying systems in Biology and Cognitive Science*, Oxford, Pergamon Press, 1991.

[18] F. Lahaie, J.R. Grasso, P. Marcenac, S. Giroux, "Self-organized criticality as a model for eruptions dynamics: validation on Piton de la Fournaise volcano, Réunion", *Comptes-Rendus de l'Académie des Sciences de Paris (CRAS)*, France, Tome 323, Série II.a, pp. 569-574, 1996.

[19] C.G. Langton, "Computation at the edge of chaos: phase transitions and emergent computation", in *emergent computation*, MIT Press, Cambridge, MA, pp 12-37, 1990.

[20] P. Marcenac, S. Leman and S. Giroux, "Cooperation and Conflicts Resolution in

MultiAgent Systems", in Proceedings of the *ACM SouthEast Conference*, Tuskegee, AL, K.H. Chang and J.H. Cross eds, pp. 289-291, April 1996.

[21] P. Marcenac, "Emergence of Behaviors in Natural Phenomena Agent-Simulation", *Complexity International Review*, (3), http://www.csu.edu.au/ci/ vol3/cs964/cs964.html, 1996.

[22] P. Marcenac, "The multiAgent approach: Complex simulations that spew realistic behaviors require independent acting variables", *IEEE-Potentials*, pp. 19-23, February/March 1997.

[23] S.A. Miller, "Earthquake as a coupled shear stress - high pore pressure dynamical system", in *Geophysical Review Letter*, 1996.

[24] N. Minar, R. Burkhart, C. Langton and M. Askenazi, "The SWARM Simulation System: a Toolkit for Building Multi-agent Simulations", *internal research report*, available in http://www.santafe.edu/projects/swarm/, June 1996.

[25] M. Mitchell, D.R. Hofstadter, "The emergence of understanding in a computer model of concepts and analogy-making", in *Emergent computation*, MIT Press, Cambridge, MA, pp. 322-334, 1990.

[26] J.R. Searle, *The rediscovery of the mind*, Gallimard, 1992.

[27] C. Taylor, "Fleshing out Artificial Life II", in *Artificial Life II*, C. Langton, C. Taylor, D. Farmer, & S. Rasmussen (eds), Addison Wesley, 1991.

[28] L. von Bertalanffy, *General System Theory*, - Dunod eds, 1993.

[29] G. Weiss, "Adaptation and learning in multiagent systems: some remarks and a bibliography", in proceedings of *Adaptation and Learning in Multi-Agent Systems*, IJCAI'95 workshop, Montreal, Canada, G. Weiss, and S. Sen (eds), Springer-Verlag, LNAI vol. 1042, pp. 1-21, 1996.

A Computer-Simulation Tool for Evolutionary Systems

Pierre Marcenac

[Marcenac 98a]

Accepted Paper

To be printed in International Journal of Knowledge-based Intelligent Engineering Systems

A Computer-Simulation Tool for Evolutionary Systems

Pierre Marcenac

IREMIA, University of La Réunion, BP 7151, 97715 St Denis Messag. Cedex 9,
La Réunion, FRANCE. Tel: (+0) 02-62-93-82-84 Fax: (+0) 02-62-93-82-60
email: marcenac@univ-reunion.fr - url --> http://www.univ-reunion.fr/~marcenac

Abstract

This paper presents a simulation platform which has been implemented in Java 1.1 for the study of chaotic behavior and evolutionary processes in non-linear systems. To support the modeling of such complex systems, the platform is based on an agent-oriented approach. It has been designed for large knowledge-based systems and is well suited in an industrial context. Resulting applications are then simulated to adequately capture any behavior likely to be observed, to exhibit unknown parameters, and to emphasize complex processes which are brought in action by such bad-understood mechanisms. This approach then allows the study of macroscopic collections endowed with the potential to evolve during time.

1 Introduction

The frightening complexity of non-linear systems constitutes a challenge for computer science modeling. In complex systems, individual components make decisions in accordance with various rules, on the basis of local, rather than global, information. Computational tools should provide intelligent capabilities to simultaneously integrate in a same frame numerous and various information and to make the synthesis of events which could follow from. The boom of computer science in systems modeling during the past twenty years has greatly increased the understanding of complex systems by using virtual simulation. This technology is both proposed by researchers and industrials, and constitutes in a real research-industry transfer. From multiple experiences, simulation is now considered as ahead of theoretical approaches, and is becoming a promising tool in this framework.

From these observations, we have built an adequate agent architecture allowing to easily distribute the system complexity in tiny pieces and make the necessary mechanisms, which will interpret local interactions and compute global results, apparent and usable. The design of the architecture has been tackled with an OMT approach [10] and has led to implement three abstractions levels: a micro-level, describing atomic and determinist behaviors in simple (reactive) agents; a medium-level, arranging intermediate structures; and a macro-level, describing the whole system where emerging behavior will be observed and analyzed. At run time, interactions between micro-agents lead to a critical state which is commonly known as the most favorable state to observe chaotic behavior [1]. Such appropriate conditions then trigger the system dynamics which interpret them in higher levels of the architecture until the system provides some results. Multiagent modeling helps us to investigate such an approach. It allows us to integrate partial results in a same frame mode. From a conceptual point of view, working with agents provides knowledge divisions that help reduce a design's intricacy when tackling complex systems.

This approach has been used to develop two applications to simulate physical processes in 1995 and 1996, thanks to the first prototypal version of the platform written above Smalltalk-80. The first one investigated volcano eruptions predicting. In particular, we studied the volcano of La Fournaise in La Reunion island which is one of the most active volcanoes on Earth. A computational model with communicating agents was used, the goal being the observation of eruptions emergence through the study of local magma pressures. Simulations have been performed for two years on the application and in situ observations were remarkably reproduced [5]. The second one reports a very interesting experience dealing with the understanding of natural structures emergence by simulation. The application has been designed to emphasize the necessity of a multi-level description in order to manage self-organization. The proposed application models and simulates a system representing a break, composed of rock cells and subject to earth's crust constraints. When such cells become unstable, they release some energy and, if the amount of energy is sufficient, then propagate

the disturbance to the neighborhood leading to earthquakes. From local basic behaviors, the model we implement has reproduced the break complexity at a global scale. The adequation between synthetic data and in-situ observations of such a break helps in validating both the agent approach in modeling and associated self-organization processes emphasized [6].

The organization of the paper is as follows. Section 2 presents the objectives and challenges of the platform. Section 3 briefly describes the layered architecture we design with these aims, before section 4 details software units. Section 5 reports the implementation and working of the platform by focusing on the heart of the system.

2. Objectives of the Platform

Our "customers" are industrials and researchers who want to explore and understand the field of potential actions in a specific domain by simulation. In the domain to investigate, the question is not "what has happened?" or even "what might have happened?", but rather "what are the sufficient conditions for a given result to be obtained?". The aim of the platform is to provide a complete toolkit as a virtual laboratory for designing a large scope of dynamic systems, studying the informational structure of complex systems and providing generic interfaces to set and control the simulation.
This kind of tool is meant for researchers needing to simulate complex systems, without having to implement the whole application by their owns. In terms of software engineering issues, this objective requires genericity and reusability as a first objective of the platform design, that justifies the use of object-oriented design methodology such as OMT.
This work has been then dictated by the following factors:

- Reducing complexity to model complex systems requires appropriate structures,
- In the framework of complex systems modeling, there are no appropriate tools (especially in simulation),
- Intrinsic mechanisms of chaotic behavior are complex and not clearly identified.

3 Engineering and Abstraction Layers

Fig. 1. gives a general view of the platform which is briefly exposed in this section, before being detailed in the next ones. Horizontally, the platform is described following three software engineering layers:
The *Language* layer, describing Java 1.1 as an object-oriented language [11], which has been chosen to implement the platform. The reasons of this choice are double: first, the clarity and cleanness of the language according to the object-oriented philosophy and mechanisms (all-is-object, static and dynamic inheritance, dynamic overloading, polymorphism...) as well as utilities useful for agent's implementation such as threads; second, the safety catch when using Applets, that is small, subordonate or embeddable applications to be run and used within the context of a larger application such as a web browser for instance [9].

The *Agent* layer, adding agents capabilities to Java, that is independence, autonomy and evolution features. Independence allows the agent to process tasks without explicitly receiving the order. Actions are performed through asynchronous messages passing, that is an agent does not wait for another resource to proceed. At the same time, an agent is given an autonomous part which can control its behavior. The autonomous part of an agent determines its self-governing: an agent is able to take an initiative and exercise a flexible degree of control through its own actions, by dynamically choosing which actions to invoke. An agent also evolves during its life, thanks to relationships kept with other agents, and adapts behavior to the environment.

Finally the *GUI* Layer, hiding the complexity of languages and agents to the user, by introducing interfaces needed to build applications and derive simulations with the platform. The GUI Layer helps in integrating the platform as a toolkit and a virtual laboratory for complex systems simulation purposes. It includes a generator, providing facilities to build applications upon the platform (following the example of Computer-Assisted Software Engineering tools), and a simulator to easily derive simulation from applications. Application consists in the description of Java classes of the domain to tackle, and one simulation in a particular instanciation of previous classes, by setting appropriate values to slots. Both GUI tools propose fill-in the blanks windows to perform such tasks.

The platform is also vertically based on four knowledge abstraction layers:
The *Object* layer is the language layer. It is composed of each Java class being used or derived to implement the sub-abstraction layers. Fig. 1 illustrates some of these classes, for instances *Thread* allowing parallel tasking, *Observable* allowing demon functions to be used to, or *Vector* to dynamically manage a list of values.

The *GEAMAS* layer, corresponding to the GEneric Architecture for MultiAgent Simulation. This layer implements a computational model of agents to

match the specifications required by non-linear systems. The GEAMAS layer constitutes the heart of the platform. It includes a specific architecture (composed of a macro-agent, medium-agents and micro-agents), the description of an agent and a society model, and self-organization mechanisms. Self-organization is a dimension of complexity. It is assigned to the appearance in a specific context and in an active environment, of new structures not previously identified, and from now on irreversible within a system of interacting entities. It is therefore important to keep trace of these structures during the simulation, in order to be self-adapted throughout the experimentation, and allow the analysis of underlying processes to be more precise. Fig. 2 illustrates such a layer.

The *Application* layer can be seen as the result of the GEAMAS layer applied on a specific domain. An application is then understood as a real world translation in terms of agents following the model implemented in the GEAMAS layer. Building an application then consists in describing the real world by deriving some or all classes of the previous layer by using the generator GUIs. The Application layer is a set of classes, whose definition is provided by the generator.

Finally, the *Simulation* layer is the instance level of the application. It defines the set of instances of the previous classes. The instanciation of such classes is graphically set with the help of the simulator editor, which proposes some fill-in-the-blanks windows to set each parameter value.

The next section details the heart of the GEAMAS layer which gives birth to agents as basic units to model all real world components in the platform.

4. Architecture of the Platform

The architecture of the platform is based on *three abstraction levels*. Each successive level represents a higher level of abstraction. Each abstraction level describes a degree of knowledge complexity and applies a model of agents, through the expression of knowledge, behavior and evolution capabilities. Each level is independent, executing processes asynchronously and representing a higher level of abstraction than the one below it. Furthermore, the architecture implements the *recursion* property to greatly reduce the design of the system, by applying the same agent-model to coarse-grain and fine-grain agents. To move up and down levels, two fundamental mechanisms are introduced: *Decomposition* and *Recomposition*. Decomposition allows to transfer information to lower levels, until Recomposition allows to transfer information to higher levels.

The Macro-Level
The macro level describes a coarse-grain agent, as a related element in which global behavior will be observed. It holds on the underlying agents' organization in a society. In multiagent systems, such an organization has no reason to be if it is unable to represent interactions facilities between agents. The society is therefore organized as a network of acquaintances, managing agents in the system to match global specifications. The role of the whole system is expressed through an external behavior, and driven by the simulator interface.

The Micro-Level
The micro level corresponds to fine-grain agents. It describes the micro-level as reactive agents. Each reactive agent is described by interactions, micro-behaviors and evolution capabilities. At run time, agents will interact in a concurrent way with their surroundings. The micro-level masks the complexity of the whole system which is encapsulated in agents describing low-level and basic actions. Such agents are often called *cells* in some related areas (Biology or Medicine for instances) or *micro-agents*, in a more general way. A micro-agent is reactive as in [4]: it generates behavior without explicit representation of the domain neither global task constraints. Micro-agents just interact with their surroundings, and evolve during time by updating their internal state. They get close to each other, and form a compact group. They act in response to events that are too fine-grained to be understood (or explained) by a macro-agent. Interactions between micro-agents are thus provided by signals which do not bear semantics, their meaning depending on the interpretative ability of the receiver.

The Medium-Level
However, as we seen it before, a complex system should integrate the chaotic dimension: the critical state of the system leads to "catastrophes". Our assumptions are then to consider the critical state as assigned to particular states of micro-agents, and the catastrophe to *a spatio-temporal aggregation*, in some particular context and environment, and under specific conditions, of micro-agents taking part in the phenomenon.

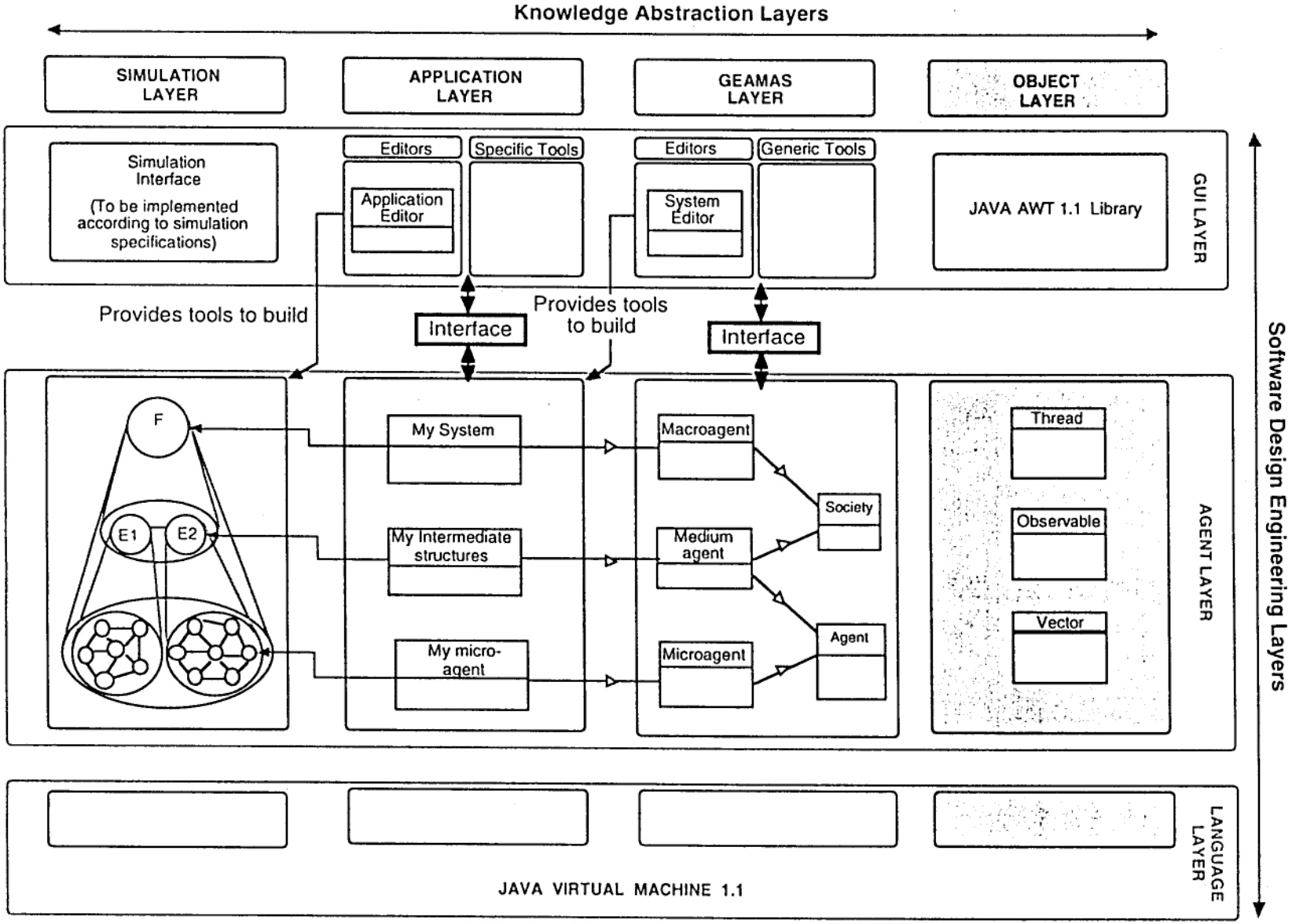

Fig. 1. A complete view of the platform

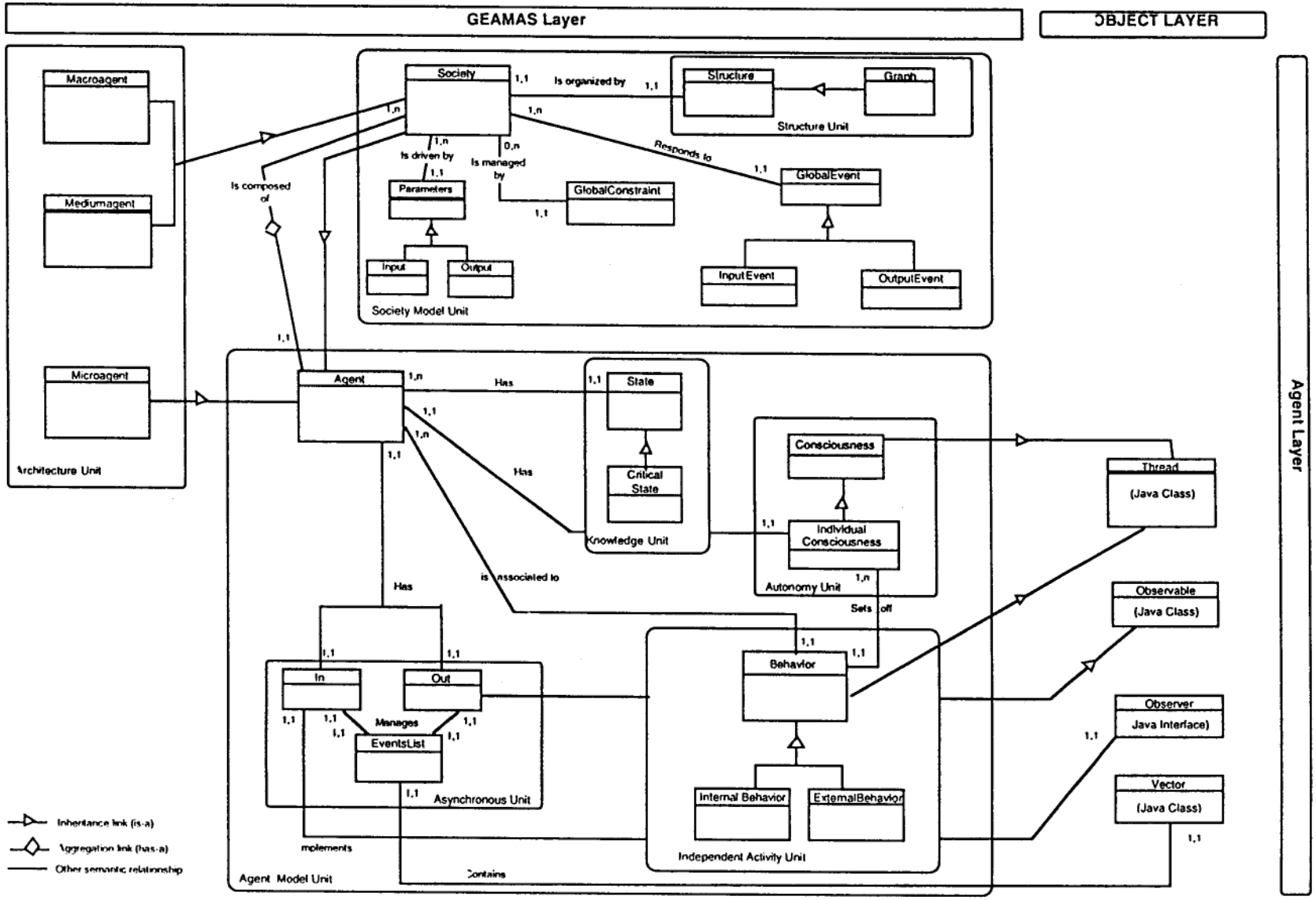

Fig. 2. Zooming on the GEAMAS layer

The result of this aggregation is then an emergent structure created by self-organization in the multiagent system.

Therefore, the main issue is to look for computational model allowing to represent and characterize self-organization, from local interactions between agents. Considering a two-levels architecture in the system makes the catastrophes modeling task difficult in the system for two reasons. On the one hand, as a micro-level is described by fine-grain agents, the organization of new structures can not be processed in such a microscopic level, which does not have enough knowledge to do so. Beyond this microscopic dimension, emergent structures can only arise in one macroscopic level, disposing of missing knowledge. On the other hand, it is often impossible for a macro-level to characterize and correctly identify the whole set of emergent structures. As the complex system is non predictable and badly understood, trying to explain emergence mechanisms at the global level of the system fails. As a matter of fact, a significant number of micro-agents could intervene in the whole emergence process, making the system too complex to be analyzed at a macro-level.

It is then skillful to introduce an intermediate grain level between the macro-level and the micro-level, embedded as sub-organizations. Such an intermediate level is called *medium level* and sub-organizations, *medium-agents*. The medium level is designed before the simulation begins by giving sub-organizations' structure and behavior. This point of view enforces the necessity of finding the most efficient abstraction when designing this level, which is totally dependent of the context tackled [2]. However, medium-agents are not dynamically pre-defined in the system because they spontaneously appear during the simulation as the result of self-organization. When self-organization arises (meaning that the system's critical state is hit), a new medium-agent will be created within the system to model the catastrophe.

This approach seems handy, because it allows to consider the catastrophe as an entire entity able to compute its own properties. Moreover, it is at the right place to distribute new computed values after the critical point in order to constraint the affected part of the system. Medium-agent's properties are computed and set when the organization has been built. Such properties are then "re-introduced" in underlying lower-level agents as constraints to apply in their own structure, this mechanism being called "*back-propagation*". By this way, a medium-agent is playing its causal role for the rest of the simulation, by constraining a part of the system.

Recursion

The recursion property constitutes an important feature to greatly reduce the intricacy of the system design, by applying the same agent model to coarse-grain and fine-grain components. Recursion expresses that a medium-agent describes a society of micro-agents which have given birth to it, and at the same time, is viewed as an agent of the society defined at the macro-level.

This duality expresses a recursive view of an agent in the architecture and unifies systems and agents: an agent can always be perceived as a multiagent system, and a multiagent system can always be viewed as a single agent. This property allows us to integrate partial results of the system in a same frame mode and to provide knowledge divisions which help to reduce a design's intricacy when tackling complex systems. An immediate advantage is the model reusability: agent and society internal structures, as well as algorithms are thus independent of the domain, and can then be reused at different levels within the architecture.

5. Implementation and Working

The three-levels architecture has been implemented in Java 1.1 following the design shown in Fig. 2. The heart of the platform is composed of the society model unit describing the global view of the application, and the agent model unit describing local behavior and interactions.

5.1. Society Model Unit

The macro-level manages agents in the system in order to match global specifications of the application and is inscribed in a model called the *society model*. The society model describes the role, the interface and the whole organization of the system: the system's role is expressed through behaviors (global events and global constraints); the interface with the external world (software, human...) describes input and output parameters for simulation needs; and the organization is managed through a structure of inter-related agents.

Input parameters govern the whole system, and describe all agents' state. In such a sense, input parameters restrict the system extent by constraining agents' state. Output parameters express the role of the system. Such parameters are used to measure the simulation results of the system outputs. They express the external behavior of the system to be observed by the user which constitutes the system's goal. They are managed by a graphical user interface described in the GUI editors of the Application layer.

The society's structure describes how agents are organized, and is then responsible for the collective of agents representing inter-agents connections (as interaction possibilities between agents). This organization is defined as the description of agents' dynamic relationships used to structure interactions within the system. The structure is then organized as a *network of acquaintances*, forming a competence network, where agents correspond to nodes and interaction possibilities (acquaintances relationships) to edges. The choice of a network as a data structure to organize agents is justified by its flexibility and its general

nature to be adapted to several situations [7]. In addition, it allows the description of some real world topologic representation, where the geometry can be implicitly represented. Indeed, the acquaintances number of each agent determines the topology, and constitutes a very interesting feature, showing how a real world could be designed as a three dimensional network. This parameter is called the agents' *connectivity*. As connections determine the ability of an agent to communicate, a part of the communication protocol is then defined by setting this parameter. This number of connections describes the influence an agent can have on another, and defines a spatial and geometric disposition of the agents. When simulation begins, each agent looks for acquaintances in the organization according to the network topology, so agents might communicate with neighbor ones.

In addition, some more information has been added in the graph. First, edges could be strengthened to take the signal intensity between two agents into account. This feature allows a certain priority of messages among agents to be expressed by the application designer. Indeed, the intensity of a signal emitted by an agent decreases as a function of the distance between agents, and agents' behavior is strongly dictated by their relative position in the topological structure. In our model, agents evolve in a world whose structure is defined by the agents' communication network, and an agent can solely interact with its immediate neighborhood. Agents' behavior and state are therefore influenced by the actions of their neighbors. When an agent acts or changes state, some of its neighbors may react more or less strongly according to the signal intensity.
Second, some agents are shown as able to receive external solicitations. This kind of agent will be first considered when an external event has to be performed by the system. In descriptive real worlds such in Physics for instance, such agents generally model the system's side.
Finally, with the same idea, some agents are shown as able to provide results, modeling for instance the opposite side of the system. This information allows the system with computing simulation results or adapting behavior.

Additional information about the organization is summarized in Fig. 3 below:

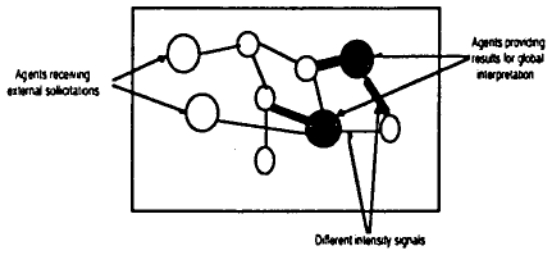

Fig. 3. Illustration of agents' organization in the society model

The behavioral part of the society is described by "global events", describing events being performed by the system. External events are first interpreted by the society which keeps control of them. As the system behavior is non-linear, this event can not be performed in the global level. The external event is therefore decomposed on underlying agents able to receive external solicitations, according to a general mechanism described below.
Some eventual global constraints can constraint the whole structure by applying specific behaviors. They are composed of general strategies and laws according to the domain tackled. For instance in a financial market, there can be a law prohibiting certain types of trades when the Dow Jones Industrial Average has declined more than 50 points from its previous close[11].

Global Working

Two fundamental mechanisms are identified: first, a set of decomposition mechanisms which allow a global input event to be distributed over the network in order to be performed by lower level agents; second, the part of the self-organization mechanisms needed to create, manage, and preserve consistency of medium-agents during phenomena emergence.

Decomposition general mechanisms

Three tasks are carried out when distributing events over the network. First, a process, named *Decomposition*, able to propagate a message to underlying agents. Decomposition allows information to be transferred in lower levels of the architecture. When receiving a Decomposition message, the agent gets information given by its society. A Decomposition message could specify that an external event is to be performed by lower level agents. This type of message helps micro-agents to be ready to perform the event. The event is thus divided up, and "sub-events" are transferred to micro-agents which are the best able to process them. Conversely, there may be laws or knowledge on a macro-level that must be observed by micro-agents, for instance, knowledge that constrains micro-agents moves. A society is at the right place to express such knowledge; it may constrain part of the state and behavior of their subordinate micro-agents, either by acting on the world structure or on individual micro-agents.
Second, a process able to find the *handling* underlying agents which can perform an event, according to agents defined as able to receive external solicitations in the organization. It defines a kind of process which is responsible for localizing the necessary agents to perform a kind of event and which remains coherent with the current goal of the society.
Finally, a *filtering function*, which receives an external event in input and provides "discretized" sub-events in output. Such a function is application-relative, as the way to transform an event in such tiny simulated events can not

[11] This rule was instituted by the New York Stock Exchange shortly after Monday, October 19, 1987 to prevent the kind of panic seen during this Black Monday.

be generic anyway. For instance, in a geophysical application such as volcano eruptions, external perturbations are caused by specific behaviors: to simulate external fluid feeding, for instance from the earth's core, magma volumes are injected within the system. When drained, the society modeling the edifice has to distribute the external injection to micro-agents modeling magma lenses. The event is divided in order to model the fluid flow here: during the simulation, if a volume of magma V is transferred to a lens, the input volume is discretized in multiple ΔVs, and an ΔV amount of magma is injected in each micro-agent. From a programming point of view, ΔVs represent time units required to re-compute the internal pressure between two messages. Indeed, micro-agents modify its state rapidly, in order to maintain the correct pressure value and be ready for the arrival of new disturbances. At the next ΔV to be injected, micro-agent is then ready to process the new computation.

Global self-organization mechanisms

When self-organization is needed to represent emergent phenomena, a global mechanism is used to create a medium-agent aggregating some micro-agents, or adding them to an existing one. If a sub-organization already exists, the underlying medium-agent managing the sub-organization is advised and the new micro-agent is integrated within the existing structure. Note that if the two micro-agents are members of two different sub-organizations, they can be integrated on both sub-organizations, to take different emergent phenomena into account. If no medium-agent is already existing, the global creation mechanism is creating a new medium-agent responsible of the emergent structure. This agent will be then populated step by step. Finally, note that such an encapsulated micro-agent can leave the structure at any time, if its state evolves again. If only one agent remains in the structure, the sub-organization is deleted.

Beyond this mechanism, back propagation involves a medium-agent to react as a feedback on each member of its structure. At the design phase of the application, the designer gives the void structure of a medium-agent defining properties which are independent of micro-agents' ones. During the simulation, when the medium-agent is entirely populated, the global mechanism computes appropriate values for the new agent's state. Setting new properties in a medium-agent becomes easy at this point, mainly because the global creation mechanism can pick up information from lower-level. This set of new properties defines a kind of constraint or restriction governing the new structure. This constraint acts as a global law for all underlying micro-agents belonging to the structure, and back-propagation is the way to inform such micro-agents. Therefore, when the emergent structure back-propagates a constraint, it forces the behavior of underlying agents by applying them some of its properties. This view is very close to reality: during an eruption for instance, each magma lens releases energy and moves into the system. The remaining energy is then low, and such micro-agents should be properly re-initialized so that the whole system behavior might be modified accordingly.

5.2. Agent Model Unit

The agent model is the most important part of the platform, as it defines an agent as the atomic entity handled in the platform.

Agent's knowledge is represented with a "state vector", from which each coordinate describes an internal property. A subset of these internal properties, called *state parameters*, are limited by *thresholds* defining the critical state. A specific behavior is then associated with each threshold. Thresholds and state parameters are set during the designing phase of the application. The agent's stability is defined as a satisfaction state with regards of its role. An agent is then stable when state parameters values do not reach the thresholds.

Agent's behavior represents what the agent is able to do during its life. In our model, the agents' behavior can be internal or external. Internal behaviors allow agents to work without external solicitations. They describe a life-cycle of the agent and perform actions continually by updating the state vector. Internal behaviors ensure the independent part of the agent. For this reason, agent's behavior should be implemented as a thread (in the Java sense of it, that is a parallel process).

External behaviors are provided by the agent when receiving external solicitations (such as events). For instance, one of the scenarios for such an external behavior can be: (1) update the state vector, (2) compute the rest of constraint to propagate in neighborhood, (3) choose agents to which propagate the event and (4) really propagate the event in chosen agents.

The asynchronous part of the agent is derived from the use of two mailboxes: *In* and *Out*. The *In* class stores input messages that should be performed by the agent ordered by intensity. The *Out* class stores each message to be propagated in the agent's neighborhood after being locally performed.

The mechanism used to set the appropriate behavior in a neighbor agent is not given by a traditional message sending, but is based on the observer mechanism in Java. Each *In* mailbox observes the agent's acquaintances *Out* mailboxes. When an object B wants to propagate an event to its neighborhood, it leaves the message in the *Out* box. The *Out* box is observed by the *In* box of all acquaintances (Agent A for instance), which automatically catches the message. This mechanism is near of those of demon functions used in knowledge-based systems in the early 1980's. Figure 4 below illustrates such a mechanism:

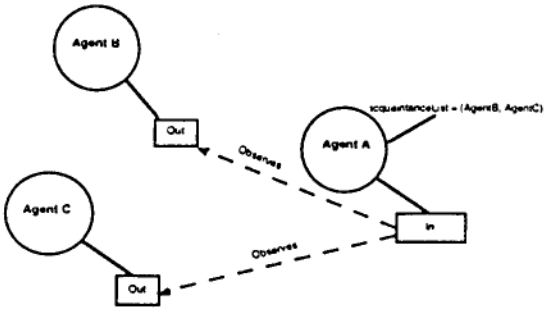

Fig. 4. Java observers and asynchronization

The autonomous part of the agent is managed by a kind of meta-control which controls the agent's behavior and determines which action to invoke. This mechanism allows to keep room for decision making, since agent's behavior is dynamically self-adapted during the simulation. This characteristic then authorizes an agent to reproduce adequate behaviors, that is behaviors the best adapted to the environment and to the current situation. Such a mechanism is called *Consciousness*. As it constitutes the over control of the agent, this process should be independent itself in order to be able to stop the agent's current behavior if some conditions change. this is why *Consciousness* should be an independent process and inherits from Thread.

Local Working

Recomposition is the "bottom-up" mechanism which is at the root of emergent behaviors of the system. It transfers information on the agent's stability from micro-level to macro-level. The values of state parameters evolve during the simulation, based on multiple local interactions between micro-agents. When one threshold is reached, the agent is assumed to be unstable, and information is then transferred to the agent's society with a Recomposition message. Therefore, a Recomposition message is provided by micro-agents to alert the society that something unusual is happening. Shifting from the micro-level up to the upper-level, the society collects data on microbehaviors and combines them to determine macrobehavior and then adapts itself to the situation. Recomposition then requires the society some abilities to interpret information at low-levels. A higher level behavior then emerges from such an ability. In such cases, the dynamic of low-level agents governs emerging behaviors.

Local self-Organization mechanisms

Agents interact according to the organizational structure described in the macro-level. If some agents are looking for cooperation to act together in a common purpose, on the contrary others get clash. By this way, some groups of agents will be self-organized from interactions, forming new organizational structures.

A local detection mechanism, distributed over micro-agents, is locally implemented to detect and identify an effective cooperation between interacting agents leading to an emergent phenomenon. At the time an interaction occurs, an agent A alerts an agent B to look for its intentions. Then, a local observer mechanism compares agent A's intentions with those of agent B. As the similarity is detected, a cooperation is then established, and both agents get organized for a common purpose. From this self-organization, locally emerges an organizational structure that has never been described in the system before. This structure is semantically based on similarity between agents [2].

When the phenomenon stops, a mechanism in charge of ending the self-organization process examines the neighborhood to look at potential agents not enough stable (and able to propagate again the phenomenon). This mechanism is usually called *observer*. The end-detection mechanism consists in computing the number of micro-agents remaining unstable. Thus, when the counter becomes nil, the end of propagation is detected. In traditional multiagent systems, the programmed observer is generally implemented as a global loop inserted over the multiagent system or as a part of the society, to inspect the stability of underlying agents forming the structure. In our architecture, the programmed observer is distributed and locally defined, to remain close to the agent paradigm.

6. Concluding remarks

This paper has presented a generic platform to develop simulation applications for non-linear systems. The complexity of the systems tackled needs to consider the result of the program execution as provided by a set of interactions among agents. Knowledge divisions which help to reduce a design's intricacy when tackling complex systems emphasizes the use of the agent paradigm to represent knowledge.

The twisting specifications of the platform involve to represent complexity in individual elements, the system's dynamics as local interactions between agents, and the result of simulation as the emergence of phenomena by interpreting such local interactions. These challenges justify the need of an OMT approach to conduct a strong analysis and design. This software engineering point of view has led the project team to develop a specific architecture allowing to distribute the complexity in independent and autonomous agents, as well as the focusing of a model of both agent and society.

This approach was proved to be very helpful in understanding and analyzing the behavior of geophysical simulations in life-sized contexts. The global system behavior has been reproduced from a model of very simple micro-agents' behavior. This global behavior was unattainable up to there with more classical approaches. The architecture then validates both the methodology and the approach for complex systems modeling and simulation. We are now convinced that the architecture is appropriate for other natural systems simulation exhibiting

similar behavior, and we actually experiment the approach in the social behavior framework, where challenges are economical rather than ecological [3].

Acknowledgments

It is a pleasure to acknowledge the support of the Institut de Recherche En Mathématiques et Informatique Appliquées (IREMIA) for the simulating intellectual environment it provided to me, as well as its financial support of this project. I am specially indebted to R. Courdier and S. Calderoni for implication in this work, J.C. Soulié for Java programming and S. Giroux for Smalltalk advice when experimenting with the first version of the platform.

References

[1] Bak, P. Tang, C. and Wiesenfeld, K. Self-Organized criticality: An explanation of 1/f Noise, Physical review Letter 59,4 (1987), 381-384.

[2] Calderoni, S. and Marcenac, P. Emergence of Earthquakes by MultiAgent Simulation in Proceedings of European Simulation Multi-Conference, ESM'97, Istambul, Turquey, June 1997, to be printed.

[3] Conte R. and Gilbert N. Introduction: computer simulation for social theory. in Artificial societies - the computer simulation of social life, UCL Press, London, 1995, 1-18.

[4] Ferber, J. Reactive Distributed Artificial Intelligence: Principles and Applications, in Foundations of Distributed Artificial Intelligence, N. Jennings eds, North-Holland, 1994.

[5] Lahaie, F. Grasso, J.R. Marcenac, P. and Giroux, S. Self-organized criticality as a model for eruptions dynamics: validation on Piton de la Fournaise volcano, Réunion, Comptes-Rendus de l'Académie des Sciences de Paris 323, II.a (1996), 569-574.

[6] Marcenac, P. and Calderoni, S. Self-organization in agent-based simulation, in proceedings of Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'97, Ronneby, Sweden, May 1997, to be printed.

[7] Marcenac, P. Emergence of Behaviors in Natural Phenomena Agent-Simulation, Complexity International Review 3, http://www.csu.edu.au/ci/vol3/cs964/cs964.html, 1996.

[8] Marcenac, P. The multiagent approach: Complex simulations that spew realistic behaviors require independent acting variables, IEEE-Potentials 2,3 (1997), 19-23.

[9] Neimeyer, P. and Peck, J. Exploring Java, O'reilly eds, 1996.

[10] Rumbaugh, J. Blaha, M. Premerlani, W. Eddy F. and Lorensen, W. Object-Oriented Modeling and Design, Prentice Hall Int. Eds: London, England, 1991.

[11] Zukowski, J. Java AWT reference 1.1, 1074 pages, 1st edition, O'Reilly,eds, 1997.

Un modèle multi-agents de l'apprenant

Stéphane Leman, Pierre Marcenac, Sylvain Giroux

[Leman et al. 96]

Article paru dans

Sciences et Techniques Éducatives, Hermès, Vol. 3, N°4, Pages 465-483, 1996.

Un modèle multi-agents de l'apprenant

Stéphane Leman* — Pierre Marcenac* — Sylvain Giroux**

** IREMIA - Université de La Réunion*
B.P. 7151 - 97715 Saint-Denis Messagerie. Cedex 9
La Réunion - FRANCE
E-mail : leman@univ-reunion.fr

*** Centre de recherches LICEF, Télé Université*
Montréal, Québec, Canada

RÉSUMÉ. *Notre objectif est de construire un modèle de l'apprenant avec une approche multi-agents. Les connaissances de l'apprenant sont représentées par plusieurs niveaux de granularité. À chaque type de connaissance correspond un type d'agent. La description interne, l'organisation dans des graphes de raisonnement, ainsi que la modification dynamique de ces agents au cours de la résolution des problèmes par l'apprenant sont décrits dans la première partie de cet article.*
Par ailleurs, lorsque l'apprenant effectue un raisonnement plus complexe, mettant en jeu plusieurs agents, ce raisonnement est reconnu comme sous-ensemble d'un raisonnement expert et est modélisé dans le système par un agent complexe. L'algorithme d'expertisation permettant cette reconnaissance et cette modélisation des raisonnements est détaillé dans la seconde partie de l'article.

ABSTRACT. *Our goal in this paper is to propose a generic method to build a student model with a multi-agent approach. Student knowledge are represented in the system with many granularity levels. Agent internal structure, reasoning graph organisation and dynamic agent evolution are describe in the first part. After, when student build a new raisoning with many agents, this reasoning is recognizing and modeling by our system. The 'expertisation' algorithm is detailled in the second part of this paper.*

MOTS-CLÉS : *modélisation du raisonnement, modèle de l'apprenant, systèmes tuteurs intelligents, systèmes multi-agents.*

KEY WORDS : *reasonning modeling, student model, intelligents tutoring systems, multi-agent systems.*

1. Introduction

Cet article s'intéresse à la construction du modèle de l'apprenant d'un système tuteur intelligent. Le principal apport du travail est de proposer une solution à la modélisation d'un apprenant en phase de résolution de problèmes. Cette modélisation de l'apprenant et de ses raisonnements est essentielle dans le cadre d'un système tuteur intelligent car elle permet de personnaliser l'enseignement par le choix de stratégies tutorielles adaptées [GRE 94]. La construction du modèle de l'apprenant reste un problème difficile et complexe [NIC 88], [HOL 94]. Il s'agit de trouver une représentation efficace des connaissances et des raisonnements de l'apprenant. Les raisonnements de l'apprenant n'étant ni figés dans le temps ni nécessairement uniques, le système de représentation doit permettre à la fois une évolutivité et une représentation simultanée de plusieurs raisonnements effectués en parallèle. Par l'autonomie des différents agents qui les composent et par leurs possibilités d'évolutions, les systèmes multi-agents offrent de nombreux avantages pour modéliser les raisonnements d'un apprenant et cet article présente des éléments de solutions originaux au problème de modélisation de l'apprenant grâce à l'approche multi-agents.
Les connaissances de l'apprenant sur le domaine sont vues comme un sous-ensemble des connaissances de l'expert, appelées *connaissances expertes*. Les connaissances expertes sont partagées entre connaissances statiques et connaissances dynamiques représentées dans notre système par des agents de types S et D. Les deux types de connaissances se lient entre eux pour former des raisonnements complexes. Au cours de la résolution des exercices proposés, l'apprenant enrichit ses connaissances. Cette évolution est prise en compte dans notre modèle par la gestion de valeurs de *croyances* attribuées aux agents. Lorsque les actions de l'apprenant ne correspondent plus exactement à une description donnée par l'expert, le système tente alors de reconnaître le raisonnement de l'apprenant comme une portion du raisonnement de l'expert. La modélisation de ce raisonnement de l'apprenant est ensuite

introduite dans le système de représentation par un agent particulier qui représente un raisonnement complexe de l'apprenant suffisamment maîtrisé pour être utilisé de façon implicite.

L'article comporte deux parties : la première décrit les composants et l'architecture générale du modèle de l'apprenant, alors que la seconde montre comment le modèle se construit dynamiquement pendant la résolution des problèmes par l'apprenant.

2. Architecture générale

2.1. Un modèle multi-agents

La structure générale du modèle se base sur les idées émises dans [SEL 87] et la construction du modèle de l'apprenant repose sur un algorithme de recouvrement issu de [CAR 77], mais cette fois distribué, car étendu aux systèmes multi-agents. Dans ce type de système, les agents coexistent : ils possèdent chacun des connaissances propres et une autonomie de fonctionnement. Le comportement global du système multi-agents est la résultante des actions propres des agents et des communications qu'ils établissent entre eux. Le modèle multi-agents proposé ici permet d'apporter certains avantages liés à la représentation distribuée des connaissances dans le domaine des systèmes tuteurs intelligents, où les approches de type multi-agents restent peu nombreuses bien que prometteuses [LEV 94]. Cette représentation permet de réduire la complexité des expertises représentées, puisque celles-ci sont réparties dans des agents autonomes, chacun en possédant une petite partie. Enfin, les avantages obtenus grâce à cette approche concernent également les deux points suivants : évolutivité du modèle au cours des différentes sessions et examen d'hypothèses parallèles lors de la construction du modèle.

2.1.1. Évolutivité

Pour conserver sa pertinence, le modèle de l'apprenant doit être évolutif. En effet, les connaissances de l'apprenant évoluent à la suite des différentes sessions d'utilisation du système. C'est d'ailleurs là que réside l'objectif primordial d'un système tuteur. Chaque action de l'apprenant est susceptible de faire évoluer ses connaissances. Ainsi, le même modèle utilisé en début de session ne sera plus aussi fidèle à la fin de celle-ci [IKE 93]. De ce point de vue, un modèle *idéal* doit être modifié en temps réel à chaque action de l'apprenant. L'utilisation d'une approche multi-agents fournit dans ce cas une alternative intéressante. En effet, dans notre modèle, chacune des interactions entre l'apprenant et le système tuteur est prise en compte localement par l'agent concerné et la mise à jour du modèle de l'apprenant se fait ainsi en temps réel sans difficulté.
Un autre aspect de l'évolutivité du système concerne la modification du type des connaissances de l'apprenant. Au fur et à mesure que l'apprenant progresse dans la maîtrise de l'expertise, son comportement se rapproche de celui d'un expert. Il utilise de plus en plus de raisonnements complexes. La reconnaissance de ces raisonnements ainsi que leur modélisation est cruciale : chaque raisonnement complexe maîtrisé par l'apprenant correspond à une élévation de l'expertise globale maîtrisée par celui-ci. Une représentation multi-agents permet de modéliser efficacement cette évolution des connaissances de l'apprenant par agglomération d'un ensemble d'agents simples en un seul agent plus complexe.

2.1.2. Examen d'hypothèses parallèles

L'apprenant est une personne humaine dont le comportement n'est pas nécessairement linéaire et cohérent. Aussi, avant d'avoir la certitude de l'intention de l'utilisateur, il est nécessaire d'examiner en parallèle plusieurs hypothèses sur le raisonnement de l'apprenant. Pour être efficace, un modèle de l'apprenant devra prendre en compte ces hypothèses différentes voire contradictoires [LEM 96]. L'utilisation d'un modèle de l'apprenant distribué dans un système multi-agents apporte une solution intéressante pour ce type de problème, puisque par nature les systèmes multi-agents permettent le traitement d'informations en parallèle. Plusieurs possibilités de raisonnement de l'apprenant peuvent ainsi être examinées en parallèle par les agents du système. Le raisonnement effectivement choisi par l'apprenant est reconnu de façon dynamique alors que les autres solutions envisagées sont abandonnées [LEM 95].

2.2. Le modèle d'agent

Le coeur du travail consiste en la définition du modèle d'agent qui spécifie les connaissances, les comportements et les interactions de chaque agent, ainsi que l'évolution dynamique de ses connaissances et de ses comportements au sein de la société. La composition interne des agents est définie par trois caractéristiques essentielles : connaissances, accointances et croyances.

2.2.1. Connaissances

Dans les systèmes tuteurs intelligents, la représentation des connaissances, que ce soient les connaissances à enseigner ou les connaissances maîtrisées par l'apprenant, est l'un des problèmes primordiaux. Dans notre modèle, la représentation des connaissances expertes et des connaissances de l'apprenant est effectuée de manière similaire, en utilisant des agents modélisant chacun une part de l'expertise globale.
Trois types de connaissances sont définies : les *connaissances statiques* qui décrivent les concepts et les *connaissances dynamiques* qui décrivent le savoir-faire dans le domaine. Les connaissances statiques sont représentées par des agents de type **S** et les connaissances dynamiques sont représentées par des agents de type **D**. Ces deux types d'agents sont qualifiés d'agents primitifs car ils modélisent un granule de connaissance indivisible. La connaissance modélisée par un agent correspond à une partie indivisible de l'expertise à enseigner. La granularité de cet agent est fixée conjointement par un expert du domaine et un pédagogue, en fonction du niveau supposé de l'apprenant et du niveau d'expertise à atteindre. Le troisième type de connaissances concerne l'enchaînement du savoir et du savoir-faire dans un *raisonnement* et exprime la capacité à résoudre le problème. Cette modélisation des raisonnements sera abordée dans la section 4.
L'ensemble des agents du système, communiquant entre eux, représente ainsi l'expertise globale. Cette représentation des connaissances sous forme d'un ensemble d'agents n'est pas nouvelle, puisque c'est ainsi que Minsky présentait sa célèbre société de l'esprit [MIN 85]. Toutefois, la modélisation d'une expertise complexe au travers d'un ensemble d'agents communiquants est un problème difficile. La recherche d'une démarche de décomposition a été abordée dans [LEM 93] dans le cadre d'une expertise de cryptarithmétique. Il est néanmoins impossible d'affirmer qu'il existe une démarche pouvant s'appliquer à toutes les expertises.
Pour illustrer cette caractéristique des agents, prenons l'exemple de la géométrie euclidienne : la connaissance d'un agent de type statique représente la maîtrise d'un concept comme par exemple un point ou une droite ; la connaissance d'un agent de type dynamique modélise la maîtrise d'un théorème ou d'une construction géométrique.

2.2.2. Accointances

Dans un système multi-agents, chaque agent ne possède que des informations locales. Ainsi, en ce qui concerne les communications, un agent ne peut s'adresser qu'aux agents dont il connaît l'existence : ses accointances. Les communications se font uniquement par envoi de messages asynchrones et il n'y a aucun mécanisme de synchronisation entre les agents.
Dans notre modèle, les agents n'ont qu'une représentation partielle de leur environnement. Ainsi, chacun des agents du modèle possède, au plus, trois accointances. Les accointances d'un agent représentent l'ensemble des pères, l'ensemble des fils et le sommet du raisonnement auxquels l'agent appartient. Hormis ses accointances, un agent ne peut pas communiquer directement avec un autre agent du système.

2.2.3 Croyances

L'introduction de valeurs de croyances dans notre modèle relève de plusieurs observations :
1) La maîtrise d'une connaissance par l'apprenant ne peut être représentée de façon "booléenne". Ainsi chaque connaissance de l'apprenant est modélisée par un agent de type S ou D avec une certaine valeur de croyance représentant le degré de maîtrise de l'apprenant pour cette connaissance.
2) Lors de la résolution d'un problème, plusieurs solutions envisageables n'ont pas nécessairement le même poids en terme de valeur pédagogique de l'exercice. Chacune des solutions se voit donc attribuer une valeur de croyance relative par rapport aux autres solutions. Cette valeur propre à chaque solution est répercutée au niveau de tous les agents qui la composent.
3) Dans un même raisonnement, toutes les étapes n'ont pas la même importance. Ainsi certaines étapes représentées par des agents ont plus d'importance que d'autres. Par conséquent, chaque agent se voit donc attribuer une valeur de croyance pour représenter son importance relative dans le raisonnement.

Pour plus de précisions dans la représentation des connaissances, trois valeurs de croyances sont ainsi introduites dans le système au niveau de chaque agent :
1) Croyance **propre** (Cp) : C'est la valeur de confiance attribuée à un agent et qui indique le degré de maîtrise par l'apprenant de la connaissance que représente l'agent.
2) Croyance **globale** (Cg) : Elle représente l'importance relative de la solution à laquelle appartient l'agent par rapport aux autres solutions possibles.
3) Croyance **relative** (Cr) : A l'intérieur de la solution, elle mesure l'importance relative de l'agent dans la solution.

Les valeurs initiales de Cg et Cr sont attribuées par des experts du domaine qui définissent les différentes solutions aux problèmes. Au départ, la valeur attribuée à Cp est indéterminée. Ensuite cette croyance évolue en fonction des actions effectuées par l'apprenant.
Les agents S et D permettent de représenter la maîtrise des connaissances par l'apprenant sur le domaine enseigné. Cette représentation des connaissances sur le domaine constitue l'ébauche de la modélisation d'un apprenant, mais les connaissances statiques et dynamiques prises individuellement ne suffisent pas pour résoudre un problème. Aussi, pour une utilisation plus efficace de cette représentation, il est fondamental de pouvoir modéliser l'enchaînement des savoir-faire de l'apprenant, ce qui constitue le raisonnement. Le raisonnement est représenté dans notre modèle par une société.

2.3. Le modèle de société

Si les agents modélisent les connaissances primitives, une société d'agents représentent un raisonnement utilisé pour résoudre un problème. Les raisonnements sont initialement donnés par des experts qui décomposent ainsi le problème général en une succession de sous-problèmes qui peuvent être résolus par l'apprenant. Pour représenter ces raisonnements, les agents S et D sont organisés suivant un *graphe de raisonnement* qui décrit l'ensemble des connaissances à enchaîner pour résoudre un problème posé. Un graphe de raisonnement est un graphe hiérarchique dans lequel chaque étape de la progression dans la résolution du problème est signifiée par l'un des noeuds du graphe. A chacun de ces noeuds correspond un agent S ou D du système.

En général, pour la résolution d'un problème proposé à l'apprenant, il existe plusieurs solutions. Pour chacune de ces solutions, l'expert propose un sous-graphe de raisonnement correspondant. Le système est alors composé de plusieurs sous-systèmes multi-agents correspondant aux différentes possibilités de raisonnement. Lors de la résolution du problème, l'apprenant ne suit pas forcément un seul raisonnement, il peut ainsi commencer plusieurs solutions possibles. Chacune des actions de l'apprenant est traitée en parallèle dans les sous-systèmes multi-agents. De cette façon, la solution choisie par l'apprenant n'est pas prédéfinie dans le système. Les solutions sont examinées en parallèle et celle qui est effectivement choisie par l'apprenant soit reconnue, alors que les autres solutions entamées n'aboutissent pas nécessairement.
Par ailleurs, la société d'agents n'est pas figée. Les accointances entre les agents qui définissent la structure du graphe de raisonnement peuvent évoluer de façon à intégrer de nouveaux agents correspondant à des nouveaux raisonnements de l'apprenant. Cette évolution se fait au travers d'un algorithme implanté au niveau de la société : l'*expertisation* (Cet algorithme sera détaillé en section 4).

3. Évolution dynamique des agents

Pour la construction dynamique du modèle, nous nous plaçons dans l'hypothèse de travail suivante : l'apprenant travaille à la résolution d'un problème au travers d'une interface informatique, appelée l'application hôte. Pendant son travail, il est observé grâce un système qualifié d'épiphyte, c'est-à-dire un système qui se greffe sur une application existante, l'hôte, de façon à l'espionner sans en modifier le fonctionnement.
L'évolution dynamique des agents a deux aspects : d'une part, un agent peut être modifié par une sollicitation extérieure correspondant à une action de l'apprenant ; d'autre part, un agent peut subir des modifications de son état par l'intermédiaire d'autres agents du système, selon un mécanisme qualifié de *propagation*.

3.1. Évolution par sollicitation extérieure

Initialement, les connaissances de l'apprenant sur le domaine enseigné sont complètement inconnues dans le modèle. On ne dispose pas d'informations a priori sur le niveau de l'apprenant. De ce fait, le système est alors constitué d'un ensemble d'agents correspondant aux différentes solutions expertes, mais dans lequel tous les coefficients de croyances propres ont des valeurs inconnues.
Lorsque l'apprenant effectue une opération, celle-ci modifie le modèle de l'apprenant. Le modèle de l'apprenant est alors construit selon un algorithme de recouvrement dont le motif est un agent S ou D. En effet, une action de l'apprenant correspond à l'activation d'une connaissance qui est reconnue par l'agent (ou les agents) la modélisant. La valeur de croyance propre de ces agents est alors modifiée selon une loi de mise à jour. Ainsi, toutes les actions de l'apprenant sont traitées dynamiquement par le système pour construire, au cours de la résolution par l'apprenant de l'exercice proposé, le modèle de l'apprenant.
La définition d'une loi de mise à jour de la valeur de croyance propre d'un agent est un problème complexe. Cette loi modélise le renforcement des connaissances par répétition et le réajustement des valeurs de croyances à chaque nouvelle utilisation de l'agent, c'est-à-dire à chaque sollicitation par l'apprenant de la connaissance modélisée par cet agent. Pour un agent de type S, chaque fois que l'agent est sollicité dans des contextes différents, sa valeur de croyance propre (Cp) est renforcée. On suppose en effet que l'apprenant maîtrise de mieux en mieux cette connaissance. Pour un agent de type D, la nouvelle valeur de croyance propre est la résultante de l'ancienne valeur et du résultat de l'évaluation par le système de la maîtrise de la connaissance. Par exemple, si l'apprenant doit fournir le résultat d'un calcul, l'évaluation de la valeur de croyance pourra se faire en calculant la différence entre le résultat fourni par l'apprenant et le résultat réel calculé par le système.

Dans les deux cas, après la mise à jour de sa valeur de croyance propre, l'agent va diffuser cette valeur aux agents qui modélisent la même connaissance dans les différents graphes de raisonnement. Cette diffusion se fait de proche en proche et va aboutir à une harmonisation des différentes valeurs de croyances. La valeur de croyance propre des agents évolue ainsi en permanence au fur et à mesure de la progression de l'apprenant dans la résolution du problème.

3.2. Évolution par propagation

Afin de conserver la cohérence du système, il est impératif qu'à chaque fois qu'un agent modifie sa valeur de croyance propre, il informe les autres agents du système qui modélisent la même connaissance, afin que ceux-ci mettent à jour leurs propres croyances. Comme l'agent ne les connaît pas directement, il le fait par l'intermédiaire de ses accointances. Le message de mise à jour circule ainsi dans le système en suivant le graphe de raisonnement. Lorsque ce message parvient à un agent modélisant la même connaissance que l'agent émetteur, celui-ci prend en compte l'information pour modifier sa propre valeur de croyance et diffuse alors lui aussi cette nouvelle valeur.
La nouvelle croyance propre de l'agent dépend alors de son ancienne croyance (Cp1), de l'importance relative de sa croyance globale (Gg1) et de la croyance globale (Cg2) et propre (Cp2) de l'autre agent. La nouvelle valeur de croyance propre peut être formulée par :

$$Cp = \frac{Cg1Cp1 + Cg2Cp2}{Cg1 + Cg2}$$

Néanmoins, cette propagation des valeurs de croyances pose le problème de l'équilibre et de la cohérence des informations.

Étudions par exemple le cas simplifié suivant :
Deux agents 1 et 2 représentent la même connaissance utilisée dans des raisonnements différents. Pour l'agent 1, la croyance propre est de 80% et la croyance globale de 30%. A la suite d'une action de l'apprenant, la croyance propre de l'agent 2 devient 60% avec une croyance globale de 70%.
Cet agent 2 communique, par l'intermédiaire des accointances successives, sa nouvelle valeur de croyance propre au premier agent qui recalcule alors sa propre valeur : Cp1=66%.

L'agent 1 communique cette nouvelle information à l'agent 2 qui se doit alors de calculer à nouveau sa nouvelle valeur de croyance propre : Cp2=61.6%.

Cette nouvelle valeur de Cp2 est alors elle aussi diffusée par l'agent 2 et l'agent 1 modifie à nouveau sa valeur, ...
Il est essentiel que la propagation des nouvelles valeurs de croyances entre deux agents différents modélisant la même connaissance ne soit pas infinie. Pour cela, il convient de démontrer la convergence des deux valeurs de croyances vers une même valeur λ.

La démonstration de cette convergence peut être établie facilement en étudiant la suite définie ci-dessous qui représente les valeurs successives de croyances propres des agents 1 (terme pair) et 2 (terme impair) [LEM 96]. Ainsi U_2 correspondra à la valeur de croyance propre de l'agent 1 après avoir reçu l'information de l'agent 2.

$$\begin{cases} U_0 = Cp1 \\ U_1 = Cp2 \\ U_{2n} = \dfrac{Cg1}{Cg1+Cg2} U_{2n-2} + \dfrac{Cg2}{Cg1+Cg2} U_{2n-1} \\ U_{2n+1} = \dfrac{Cg2}{Cg1+Cg2} U_{2n-1} + \dfrac{Cg1}{Cg1+Cg2} U_{2n} \end{cases}$$

Les valeurs de croyances propres de deux agents convergeant vers une même valeur, la cohérence des informations de notre système est donc assurée. Par ailleurs, pour limiter les échanges de messages, nous proposons les hypothèses suivantes : un agent dont la nouvelle valeur de croyance propre différera très faiblement de son ancienne valeur (moins de 3%) ne propagera pas cette nouvelle valeur aux autres agents du système et les valeurs de croyances propres des agents sont arrondies aux pourcentages entiers les plus proches. Avec ces hypothèses et en reprenant l'exemple décrit ci-dessus, les valeurs de croyances finales des deux agents, convergeant *théoriquement* vers 62,3%, sont Cp1=66% et Cp2=62% et seulement deux messages ont été échangés :

A2 -> A1 Cp=60 Cg=70
Pour A1, Cp1=80x30+60x70=66
A1 -> A2 Cp=66 Cg=30
Pour A2, Cp2=60x70+66x30=62
Cp2 passe alors de 60 à 62. La différence est inférieure à 3%, on ne diffuse plus la nouvelle valeur.

Il est important de remarquer qu'il y a également convergence des valeurs de croyances dans le cas où plus de deux agents sont impliqués. Toutefois, dans le cas où plusieurs agents sont concernés, il est cette fois impossible de déterminer l'ordre dans lequel les messages sont échangés, et les valeurs de croyances des différents agents concernés ne sont donc pas déterministes à chaque instant.

Soit l'exemple suivant où deux agents A1 et A2 possèdent une croyance Cp1=Cp2=80. Un troisième agent A3 reçoit comme nouvelle valeur de croyance Cp3=60, les messages échangés peuvent être :
Soit au départ[12] :
A1,80,60 - A2,80,60 - A3,60
On ne peut pas déterminer qui de A1 ou A2 va traiter le message en premier. Supposons que ce soit A1 (les deux agents jouent dans cet exemple un rôle symétrique).
A1 passe de 80 à 70 et l'état des agents est alors le suivant
A1,70 - A2,80,60,70 - A3,60,70
La encore, deux choix sont possibles pour l'ordre d'exécution : A2 ou A3. Les résultats correspondants sont :
Cas A2 : A1,70,70 - A2,70,70 - A3,60,70,70
puis A1,70,65 - A2,70,65 - A3,65,70
Cas A1 : A1,68 - A2,70,65,68 - A3,65,70,68
Cas A3 : A1,70,65,68 - A2,70,65,68 - A3,68
puis finalement A1,68 - A2,68 - A3,68

[12] La notation suivante Ax,Cp,x1,x2,....xi signifie que l'agent X possède actuellement une valeur de croyance propre de Cp et qu'il doit intégrer successivement les valeurs de croyances x1,x2,....xi. (Hypothèse : les croyances générales sont égales).

Cas A3 : A1,70,65 - A2,80,60,70,65 - A3,65
Cas A1 : A1,68 - A2,80,60,70,65,68 - A3,65,68
Cas A2 : A1,70,65,70 - A2,70,70,65 - A3,65,70
etc...

Le comportement exact des agents reste inconnu, et donc il en est de même de l'évolution de la croyance propre de tous les agents. Ce problème est inhérent au parallélisme et au traitement asynchrone des messages, mais il offre l'avantage d'une plus grande souplesse dans le traitement des informations : il est assuré que les messages émis par les agents sont traités dans un temps fini.

4. Modélisation des raisonnements complexes de l'apprenant

4.1. Nécessité d'un nouveau type d'agent : le R-agent

Les exemples précédents ont montré comment s'effectuait la construction du modèle de l'apprenant dans les cas simples, c'est-à-dire lorsque l'action effectuée par l'apprenant correspondait bien à une action attendue par l'un des agents non encore activé. L'apprenant suivait, étape par étape, l'une des solutions décrite par un expert. Toutefois, il est possible que l'apprenant effectue un saut dans le raisonnement lorsqu'il maîtrise bien une partie de celui-ci. Dans ce cas, une ou plusieurs étapes de raisonnement données par l'expert peuvent être omises. Ce saut dans le raisonnement correspond à la maîtrise par l'apprenant d'un nouveau type de connaissance. Pour modéliser cet état, des agents de type **R** sont introduits.
Un agent de type R possède les mêmes caractéristiques que les agents primitifs S ou D en terme de connaissances, d'accointances et de croyances. Toutefois, la granularité de la connaissance qu'il représente est supérieure. Cette possibilité de représenter les connaissances de l'apprenant avec plusieurs niveaux de granularité apporte un avantage indéniable pour l'exploitation par des stratégies tutorielles du modèle de l'apprenant [CAL 94].
Toutefois, il n'est pas approprié de créer un agent de type R à chaque fois que l'apprenant effectue un saut dans le graphe de raisonnement. Il est au préalable important de vérifier que les connaissances sautées par l'apprenant dans ce raisonnement sont bien maîtrisées. Si tel est le cas, l'apprenant a acquis une nouvelle connaissance plus complexe que celles qu'il maîtrisait jusqu'alors, et un agent de type R doit être créé dans le modèle pour représenter cette nouvelle connaissance. Pour résoudre le problème de la validation de l'expertisation de l'apprenant, l'algorithme comprend deux parties. Dans un premier temps, il convient de reconnaître le raisonnement de l'apprenant dans l'un des graphes de raisonnement. Ensuite, et seulement dans le cas où le raisonnement est reconnu, il convient de le modéliser par la création d'un agent R.

4.2. Reconnaissance du Raisonnement

Reconnaître le raisonnement de l'apprenant revient donc à rechercher si l'apprenant a ou n'a pas effectué un saut dans le graphe de raisonnement expert. Toutefois, nous ne disposons pas au niveau des agents d'une vue globale sur les graphes de raisonnement. La détection d'un saut de raisonnement se fait donc de façon locale. Lorsqu'un agent reçoit une information correspondant à une action de l'apprenant, si cette information correspond à la connaissance modélisée par l'agent récepteur, alors celui-ci met à jour sa nouvelle valeur de croyance propre et met à jour sa vision propre de l'apprenant. Il transmet ensuite cette nouvelle information aux autres agents qui modélisent la même connaissance de façon à conserver la cohérence du modèle.
Par contre, si l'action effectuée par l'apprenant et transmise à l'agent ne correspond pas à la connaissance qu'il modélise, celui-ci consulte les autres agents pour une expertisation. Dans ce cas, la première étape consiste à reconnaître le raisonnement de l'apprenant dans l'une des solutions de l'expert. Pour cela, il faut s'assurer que l'action effectuée par l'apprenant correspond à l'un des agents situés au-dessus de l'agent demandeur dans le même graphe de raisonnement. Si tel est le cas, il faut alors déterminer l'ensemble des agents qui ont été sautés par l'apprenant et qui constituent le raisonnement.

La reconnaissance du raisonnement de l'apprenant se fait donc de façon locale, de proche en proche. L'agent demandeur communique sa demande à ses pères, qui la retransmettent à leur tour à leurs pères, et ceci tant que le sommet n'est pas atteint ou le raisonnement reconnu.
La liste des agents constituant le raisonnement de l'apprenant est construite au fur et à mesure du parcours vers le sommet du graphe de raisonnement. Si l'agent recherché est trouvé, le raisonnement est reconnu comme une portion de la solution de l'expert et la liste des agents la constituant est retournée au demandeur. Un tel cas est illustré par la figure 1 :

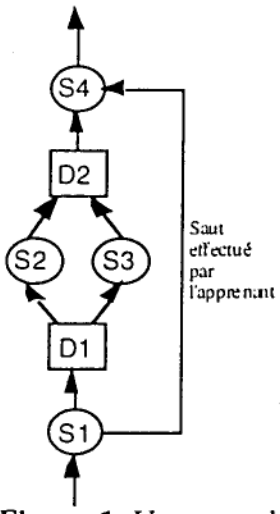

Figure 1. Un exemple de saut dans le raisonnement.

Par contre, si le sommet du graphe est atteint sans trouver d'agents correspondant à la recherche, la reconnaissance du raisonnement échoue. L'apprenant a effectué une action non reconnue dans la solution experte concernée. Cet échec de la reconnaissance du raisonnement doit alors être exploité par une stratégie tutorielle.
A l'opposé, si le raisonnement de l'apprenant est reconnu, il faut dans le cas où l'apprenant maîtrise effectivement les connaissances qu'il contient, le modéliser par un agent R. C'est l'instanciation de cet agent R, c'est-à-dire la définition de la connaissance qu'il représente et des accointances et croyances qu'il contient que nous allons aborder maintenant.

4.3. Instanciation de l'agent R

Nous sommes dans le cas où le raisonnement de l'apprenant est reconnu. Il convient alors d'examiner tous les agents pour déterminer si l'apprenant maîtrise effectivement les connaissances représentés par les agents sautés. L'agent demandeur se met en attente de réponse après avoir lancé une demande d'information sur la liste des agents sautés. Cette information est demandée directement au sommet qui la dirige alors vers l'ensemble de ses fils à l'exception de celui qui est l'ancêtre de l'agent demandeur. Un agent qui reçoit un message de demande d'informations répond s'il est concerné, c'est-à-dire si la connaissance statique ou dynamique qu'il modélise correspond à celle du demandeur. Dans ce cas, il communique ses valeurs de croyance à l'agent demandeur par l'intermédiaire du sommet. Nous sommes maintenant dans la phase descendante de l'algorithme, si l'agent n'est pas concerné, il se contente de transmettre la demande à ses fils. Ainsi, de proche en proche, tous les agents susceptibles de fournir une information sont interrogés. L'agent demandeur reçoit ainsi des informations concernant tous les agents qui modélisent les mêmes connaissances que celles des agents qui composent le raisonnement de l'apprenant. La création de l'agent R ne se fait que lorsque des informations ont été reçues sur tous les agents sautés. Il s'agit alors de traiter les informations reçues, et d'attribuer connaissances, accointances et croyances à l'agent R :

4.3.1. Calcul de la connaissance

La connaissance associée au nouvel agent R créé correspond à l'agrégat des connaissances modélisées par les différents agents que R va généraliser. La granularité de cette connaissance est donc supérieure à celle

des autres agents ce qui correspond bien à l'idée que l'on se fait des connaissances d'un expert par rapport aux connaissances d'un novice.

4.3.2. Calcul des accointances

L'agent R nouvellement créé possède lui aussi trois accointances. L'agent qui a déclenché l'algorithme sera le fils, l'agent qui a reconnu le raisonnement sera le père et le sommet de l'agent R est le même que celui de l'agent fils et de l'agent père, puisque ces trois agents appartiennent à la même solution.

4.3.3. Calcul des croyances

Comme un agent R intervient à l'intérieur d'un seul raisonnement, la croyance globale de tous les agents que généralise l'agent R est la même. La croyance globale attribuée à l'agent R est donc définie comme étant égale à cette croyance.
Par ailleurs, un agent R remplaçant une partie de raisonnement, on lui attribue une valeur de croyance relative égale à la somme des croyances relatives des agents qu'il remplace.
Enfin, pour déterminer la valeur de croyance propre associée au nouvel agent R, il convient tout d'abord calculer les croyances propres des agents $X_1, X_2, ..., X_n$ que R remplace. Il est alors aisé de calculer la valeur de Cp pour l'agent R en pondérant les différents $Cp(X_i)$ par l'importance relative des agents X_i dans la solution :

$$Cp(R) = \frac{\sum_i Cr(Xi)Cp(Xi)}{\sum_i Cr(Xi)}$$

Notre modèle de l'apprenant permet ainsi de représenter à la fois les connaissances primitives de l'apprenant par l'intermédiaire des agents S et D, mais aussi et surtout, il offre deux possibilités de représentation des raisonnements de l'apprenant avec deux niveaux de granularité différents, modélisant la maîtrise et le degré d'expertisation de l'apprenant sur une partie du domaine enseigné.

5. Éléments d'implémentation

Le système TREMMA, acronyme pour TRansfert d'Expertise avec un Modèle Multi-Agents, est écrit en Smalltalk et se base sur la plate-forme de programmation Epitalk permettant de décrire et de générer des systèmes épiphytes. Nous décrivons tout d'abord brièvement l'environnement de programmation, puis le système en lui-même et enfin nous donnons les deux exemples d'expertises avec lesquelles nous avons testé TREMMA.

5.1. Environnement de programmation

Notre environnement de programmation est la plate-forme Epitalk. Epitalk est le terme d'une succession de couches au dessus de Smalltalk-80 et notre système TREMMA hérite automatiquement d'un ensemble de propriétés issues de ces plates-formes successives.

5.1.1. Smalltalk-80

C'est un des précurseurs des langages à objets [GOL 83], dont il reste l'une des références. C'est à la fois un langage de programmation, un environnement graphique et un système d'exploitation. Il est principalement utilisé dans les applications industrielles comme un outil de prototypage d'applications. La raison essentielle du succès de Smalltalk vient de la conception du langage qui permet la réutilisation effective de classes et de méthodes existantes.

5.1.2. Actalk

Le langage Actalk écrit par Jean Pierre Briot [BRI 88] est conçu comme une couche acteur au dessus de Smalltalk-80. L'avantage d'un tel environnement de programmation est de proposer de programmer avec des acteurs tout en disposant de toutes les facilités offertes par l'environnement Smalltalk. Pour transformer les objets de Smalltalk-80 en acteurs, l'auteur utilise deux techniques de programmation, d'une part pour rendre les objets actifs et d'autre part pour transformer les communications entre ces différents objets actifs et les rendre asynchrones. Un acteur est ainsi composé de deux objets, l'un

décrivant son comportement (*behavior*) et l'autre sa boîte aux lettres (*mailbox*). Par ailleurs, une pseudo-variable *aself* permet de référencer l'acteur au niveau de son comportement.

5.1.3. ReActalk

Smalltalk-80 sert de niveau de base. Au niveau de la couche Actalk, les capacités des objets passifs de Smalltalk-80 sont redéfinies pour en augmenter l'autonomie et pour obtenir une sémantique asynchrone de la transmission de messages. Les objets passifs accèdent au statut d'acteurs. Cette approche ne permet toutefois pas les modifications dynamiques du comportement d'un acteur pris isolément. Sa vie durant, un acteur reste lié à sa classe, et par conséquent se comporte en stricte conformité avec les connaissances et comportements hérités. La couche ReActalk [GIR 93] introduit la réflexivité et complète l'autonomie en libérant les acteurs de la hiérarchie d'héritage. La réflexivité autorise les changements, même temporaires, du comportement par l'intermédiaire d'un méta-acteur.

5.1.4. Epitalk

C'est le dernier maillon de la chaîne d'évolution successive de Smalltalk-80 que nous utilisons. Epitalk [PAQ 94] permet de générer des systèmes épiphytes multi-agents, c'est-à-dire des systèmes qui se greffent sur une application existante de façon à l'espionner sans en modifier le comportement. Ainsi, tandis que l'apprenant travaille dans l'application hôte, le système épiphyte garde dans le même temps une trace de l'ensemble des interactions.
Pour implémenter notre système, les différents agents S et D sont définis comme des agents de Epitalk, c'est-à-dire qu'ils sont actifs et peuvent communiquer avec des agents avec lesquels ils sont en relation. Ces relations entre les différents agents forment un graphe isomorphe au graphe de raisonnement. Pour réaliser l'espionnage de l'apprenant, Epitalk utilise un autre type d'agent : les agents *espions*. Ces derniers recueillent les informations concernant les actions effectuées par l'apprenant dans le système hôte et les transmettent aux agents S ou D correspondant.

5.2. Le système TREMMA

La programmation de notre modèle consiste en la définition de la classe générique des agents du système décrivant les connaissances des agents comme des variables d'instances de cette classe, et en la définition des méthodes correspondant aux comportements. De même, il convient de définir l'organisation des agents au sein de la société. Les comportements et les connaissances de cette société sont décrits au travers des variables d'instances et des méthodes appartenant à la classe définissant le modèle générique de société.
Un agent TREMMA est principalement caractérisé par les connaissances propres qu'il contient. En particulier, quelle que soit l'application pour lequel il va être utilisé, un agent de notre modèle est une instance de la classe *TREMMA* et il doit posséder un nom (*name*), un identificateur unique (*id*) et un ensemble d'accointances (*accointances*). Ces différentes informations sont stockées dans le comportement de l'agent. Chaque agent possédant également une boîte aux lettres. Comme les agents TREMMA bénéficiaient de l'héritage des acteurs Actalk et agents ReActalk, une société TREMMA est une descendante des sociétés d'agents définies dans ReActalk et dont la caractéristique principale est d'être un écosystème qui peut se définir par un ensemble d'agents structuré selon un graphe organisationnel représentant le réseau des relations entre les agents. Cette notion d'écosystème, telle qu'elle existe déjà en ReActalk convient parfaitement pour représenter une société TREMMA.

L'application au transfert d'expertise utilise le modèle multi-agents générique. La programmation consiste donc à enrichir les classes génériques de TREMMA. Cet enrichissement se fait en créant les classes d'agents et en introduisant les connaissances et les comportements spécifiques au transfert d'expertise. Dans le cadre d'une application de transfert d'expertise, il est nécessaire de créer les classes correspondant aux agents S, D et R qui sont des sous-classes de *Tremma*. Toutefois, dans un souci de simplification de la programmation, il est intéressant de factoriser des comportements communs dans une classe intermédiaire *TremmaTE*.
Il convient aussi de spécifier les connaissances d'un agent : ainsi, par exemple, la variable générique *accointances* de l'agent Tremma peut être décrite comme une liste de trois variables. On a donc *accointances := (pères, fils, soc)*. D'autres connaissances spécifiques d'un agent sont introduites ici et vont dépendre du type d'agent. Par exemple, un agent D reçoit des variables *entree* et *sortie* qui représenteront les agents S qu'il va accepter en entrée et en sortie. C'est également à ce niveau que sont

implémentés les comportements génériques des agents et de la société liés au transfert d'expertise, mais pas à l'expertise elle-même. On retrouve en particulier ici tous les comportements implémentant les algorithmes de recouvrement distribué, de modification des croyances par mise à jour et par propagation et d'expertisation.

5.3. Exemples d'expertises utilisées

Deux expertises ont pour le moment été utilisées pour tester le modèle. La première concerne la géométrie et nous a permis de construire un modèle de l'apprenant avec des agents S et D, et la seconde le diagnostic d'un jeu de tarot permettant d'illustrer la construction d'un R-agent. Les applications développées autour de ces expertises sont très restreintes et ne constituent pas de véritables systèmes tuteurs mais elles permettent néanmoins de valider les hypothèses prises quant à la représentation de l'apprenant par un système multi-agents. Une validation sur une expertise plus complexe est actuellement à l'étude et devra ensuite être expérimentée sur de vrais apprenants.

5.3.1. Géométrie

L'objectif est de faire construire des figures à l'apprenant uniquement à l'aide de la règle et du compas. L'apprenant évolue dans une interface, issue de [AUB 90], lui permettant de construire point, droite et cercle et le système lui propose une construction à réaliser.

Nous nous plaçons dans le cas ou l'apprenant en est à sa première séance de travail. Tous les agents ont alors un coefficient de croyance inconnu. L'exercice proposé est celui de la construction d'un carré à partir de deux de ses points.
Voici trois solutions données par des experts en géométrie :
- Construire 1 cercle et 3 droites perpendiculaires.
- Construire 2 cercles et 2 droites perpendiculaires.
- Construire 3 cercles et 1 droite perpendiculaire.

Dans ce cas, le système multi-agents représente trois graphes de raisonnement. Chacun de ces graphes de raisonnement étant lui-même composé de plusieurs agents. Si la première action de l'apprenant est la construction d'un cercle de centre A passant par B, trois agents seront activés, puisque dans chacune des solutions, il faut construire un cercle de ce type. Par la suite, les actions effectuées par l'apprenant dans l'application de géométrie sont reconnues par le système comme concourant à l'une des trois solutions. Si après avoir construit la perpendiculaire à (AB) en A, le point D, puis le cercle de centre D passant par A, l'apprenant se trouve alors dans une situation lui permettant de terminer son carré par la construction d'un cercle de centre B passant par A ou d'une perpendiculaire. Cette double possibilité est modélisée, sans choix à priori, dans notre modèle par le fait qu'un agent D activé par l'apprenant permettra d'obtenir le quatrième sommet du carré.

5.3.2. Tarot

Il s'agit d'une expertise simplifiée sur l'évaluation d'un jeu de tarot : la méthode enseignée demande à l'apprenant d'évaluer son jeu en fonction de plusieurs critères afin de choisir son enchère. L'apprenant doit évaluer successivement plusieurs paramètres avant de proposer une annonce. Cette méthode est très didactique. Chacune des évaluations à effectuer par l'apprenant est décomposée en évaluations plus simples : ainsi l'évaluation d'un jeu consiste à évaluer les atouts et les couleurs, et l'évaluation d'une des quatre couleurs consiste en l'évaluation de la force et de la longueur de cette couleur.
Lorsque l'apprenant progresse dans la maîtrise de l'expertise, il va naturellement sauter des étapes de raisonnement. Par exemple, il pourra arriver à une évaluation globale de la couleur en sautant la décomposition force-longueur. Ainsi après une évaluation décomposée et réussie des trèfles et des carreaux, l'apprenant peut évaluer globalement les coeurs. Dans ce cas, une expertisation est nécessaire et se traduit dans notre système par la création d'un nouvel agent R correspondant à l'évaluation globale d'une couleur. Avant de créer cet agent R, il convient tout d'abord d'identifier la partie de raisonnement sauté par l'apprenant, et de vérifier que les agents modélisant cette partie de raisonnement représentent des connaissances maîtrisées par l'apprenant. Ensuite le nouvel agent R est crée et pourra être directement utilisé par la suite pour modéliser un raisonnement de l'apprenant.

6. Conclusion

Cet article propose une architecture originale de type multi-agents pour modéliser les connaissances et les raisonnements d'un apprenant.
Un modèle d'agent définit les connaissances, les comportements et les interactions de chaque agent, ainsi que l'évolution dynamique de ses connaissances et de ses comportements au sein de la société. Ce modèle d'agent spécifie les connaissances d'un agent grâce à des types d'agents (S, D ou R) ainsi que ses comportements. Un modèle de société d'agents est également défini. Il décrit la structure des agents dans un graphe de raisonnement et l'évolution de la structure par un algorithme d'expertisation.
La modélisation de l'apprenant présentée dans cet article concerne la résolution de problèmes par l'apprenant. Un transfert d'expertise ne se limite pas à la résolution de problèmes. Il faut prévoir d'enchaîner différents problèmes au sein de sessions, puis ces sessions au sein d'un cursus. Cette organisation du transfert d'expertise en trois niveaux hiérarchiques permet de montrer un autre avantage de notre modèle : son auto-similarité. En effet, les systèmes multi-agents décrits dans cet article et représentant la résolution d'un problème sont alors vus comme des agents primitifs au niveau session. Ces agents possèdent alors les mêmes caractéristiques que les agents S et D et il est donc aisé de construire de manière similaire un modèle de l'apprenant au niveau d'une session puis d'un cursus.
L'utilisation du modèle par des stratégies tutorielles est une autre prochaine étape de la recherche. Deux types d'interventions sont envisagées dans un premier temps. Une première stratégie tutorielle concerne le cas où l'apprenant ne progresse plus dans la résolution du problème (c'est-à-dire qu'il n'a ni progressé dans aucune des solutions, ni exploré aucune autre solution référencée). Dans ce cas, la stratégie consiste à proposer à l'apprenant l'étape ou les étapes suivantes possibles. Une seconde stratégie consiste à proposer à l'apprenant une aide pas à pas, en lui offrant les prochaines étapes de résolution de chacune des solutions possibles.

Références

[AUB 90] M. AUBÉ, Utilisation d'un langage orienté objets pour la compréhension de concepts de géométrie, Congrès de l'AMQ, Collège de Sherbrooke, 20 octobre 1990.

[BRI 88] JP. BRIOT, Actalk : une plateforme de modélisation de langages d'acteurs en Smalltalk-80, Rapport de Recherche, Université de Paris VI, Paris, 1988.

[CAR 77] B. CARR, I. GOLDSTEIN, Overlays, a theory of modelling for CAI, AI memo 406, MIT Press, 1977.

[CAL 94] G. MC CALLA, J. GREER, Granularity-Based Reasonning and Belief Revision in Student Models, Student Models : The key to Individualized Educational Systems, pages 39-62, Springer-Verlag, New-York, USA, 1994.

[GIR 93] S. GIROUX, Agents et Acteurs : une nécessaire unité, Thèse de l'Université de Montréal, Mars 1993.

[GOL 83] A. GOLDBERG, D.ROBSON, Smalltalk-80. The Language and its Implementation, Addison Wesley, 1983.

[GRE 94] J. GREER, G. MC CALLA, Student Models: The key to Individualized Educational Systems, Springer Verlag, 1994.

[HOL 94] P. HOLT, S. DUBS, M. JONES, J. GREER, The state of Student Modelling, Student Models : The key to Individualized Educational Systems, pages 39-62, Springer-Verlag, New-York, USA, 1994.

[IKE 93] M. IKEDA, Y. KONO, R. MIZOGUCHI, Nonmonotic Model Inference. A Formalization of Student Modeling, IJCAI'93, 13th International Joint Conference on Artificial Intelligence, Pages 467-473, Chambery, France, 1993.

[LEM 93] S. LEMAN, P. MARCENAC, Modélisation sous forme d'agents et implémentation en acteurs d'une expertise arithmétique complexe, Actes de la Journée Systèmes Multi-Agents du PRC-IA, Montpellier, France, 17 Décembre 1993.

[LEM 95] S. LEMAN, S. GIROUX, P. MARCENAC, A Multi-Agent Approach to Model Student Reasoning Process, 7th World Conference on Artificial Intelligence in Education, pages 258-265, Washington D.C. , USA, 16-19 Août 1995.

[LEM 96] S. LEMAN, P. MARCENAC, S. GIROUX, Maintien dynamique de la cohérence d'un modèle de l'apprenant par coopération d'agents cognitifs, Actes de la Journée Systèmes Multi-Agents du PRC-IA, Pages 93-103, Toulouse, 2 février 1996.

[LEM 96] S. LEMAN, TREMMA : TRansfert d'Expertise aavec un Modèle Multi-Agents, Thèse de l'Université de La Réunion, 24 Juillet 1996.

[LEV 94] S. LEVESQUE, C. FRASSON, J. GESCEI, Perspectives multi-agents des systèmes tuteurs intelligents , Actes des 2° Journées Francophones IAD & SMA, Pages 55-66, Voiron, France, 9-11 Mai 1994.

[MIN 85] M. MINSKY, The Society of Mind, Simon and Schuster, 1985.

[NIC 88] JF.NICAUD, M.VIVET, Les tuteurs intelligents : réalisations et tendances de recherches, Techniques et Sciences Informatiques, Volume 7, Numéro 1, Pages 21-45, 1988.

[PAQ 94] G. PAQUETTE, F. PACHET, S. GIROUX, EpiTalk, un outil générique pour la construction de systèmes conseillers, Sciences et Techniques Educatives, Vol 1, N° 3, Pages 305-336, Novembre 1994.

[SEL 87] J. SELF, Student models : what use are they ? , Artificial Intelligence tools in Education, proceedings of the IFIP TC3 working conference on Artificial Intelligence tools in Education, Frascati, Italie, mai 1987.

Stéphane Leman est docteur en informatique de l'Université de la Réunion depuis le 24 juillet 1996. Ses travaux de recherches, menés à l'Institut de REcherche en Mathématiques et Informatique Appliquées (IREMIA), s'intéressent aux apports d'une approche multi-agents pour les systèmes tuteurs intelligents et plus particulièrement pour la modélisation de l'apprenant. Sa thèse a principalement débouché sur la définition d'un modèle multi-agents pour la représentation d'une expertise en vue de son transfert.

Pierre Marcenac est actuellement maître de conférences à l'Université de la Réunion, où il travaille depuis 1992 à l'élaboration de plates-formes à base d'agents. Ses centres d'intérêt sont la modélisation des processus de raisonnements des apprenants en phase de résolution de problèmes et la modélisation de systèmes complexes.

Sylvain Giroux Sylvain Giroux est actuellement professeur substitut à la Télé-université, Université du Québec. Après avoir obtenu un Ph. D. en informatique à l'Université de Montréal en 1993, il a fait ses études postdoctorales à l'Université de La Réunion où il a travaillé sur l'adaptation dans les systèmes ouverts et sur la simulation d'édifices volcaniques à l'aide de systèmes multi-agents. Ses travaux de recherches actuels portent sur les systèmes conseillers épiphytes multi-agents et sur l'utilisation des hypermédias dans la réalisation de cours pour la formation à distance.