Applied Intelligence 8, 247–267 (1998)
© 1998 Kluwer Academic Publishers. Manufactured in The Netherlands.

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

PIERRE MARCENAC
IREMIA, University of La Reunion, BP 7151, 97715 St Denis Messag. Cedex 9, La Reunion, France
marcenac@univ-reunion.fr

SYLVAIN GIROUX
CIRANO, 2020 rue University, 25eme etage, Montreal, Quebec, Canada H3A 2AJ
girouxs@cirano.umontreal.ca

Abstract. This paper's object is to present the results of the GEAMAS project which aims at modeling and simulating natural complex systems. GEAMAS is a generic architecture of agents used to study the behavior emergence in such systems. It is a multiagent program meant to develop simulation applications. Modeling complex systems requires 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.

Keywords: intelligent distributed processing, complex systems modelling, multiagent systems, smalltalk, simulation applications

1. Introduction

The ability of natural phenomena to cause damage upon the earth each year has recently constituted a challenge for computer science modeling and is now 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. Simulation is relevant in this framework, because it can adequately capture any behavior likely to be observed. It is used to exhibit new parameters and to tackle their role in 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. 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 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 the 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.

Furthermore, a complex system is seen as a nonlinear system, that is a system where 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. Both deep interaction and imprecision attached to the available knowledge involve a radical change in problem solving: algorithmic methods, useful whenever a unique output is delivered for every input are doomed to failure. 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.

There is no appropriate tools to model and simulate complex systems with multiagent systems. Our proposal is to propose a framework that reduces and organizes the system complexity. The challenge is then to understand why interactions between agents 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 Langton's works on artificial life in Santa-Fe [32].

To address these issues, a layered architecture, called GEAMAS[1] is introduced. It is seen as a multiagent development platform, intended to develop simulation applications for complex systems. GEAMAS is an architecture that exploits concurrency and distributing computing as key issues of the behavior of the system.

Imagining a layered architecture is one of the possible answers to reduce the complexity by abstraction [18]. A better understanding of the complexity is then brought up through the expression of fine-grained and coarse-grained components within the system. In addition, such an architecture can provide information on how behavior could emerge out of interacting agents, 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 than the one below it. Each abstraction level describes a degree of knowledge complexity: it applies a model of agent through the expression of interaction, behavior and evolution capabilities. Each level is independent, that is executes asynchronous processes.

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 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, we have investigated complexity through the modeling of natural systems, such as those studied in geophysics, more particularly volcano eruptions. The whole application was tested by our team: more than one hundred simulations were performed during the first twelve months. The interpretation of this year's results was 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 as key issues to model complex systems. 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 architecture implementation 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. Agent as the Key Point

Our framework proposes the use of agent technology as high level tool to design complex systems. 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. The meaning of agent is hard to define and depends on the context tackled. There is no common ontology (that is no common structured set of concepts), nor real agreement on what is an agent. Recently, Franklin and Graesser propose to characterize agents by its sensing and acting capabilities [19]: "An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future".

This formal definition gave rise to much discussion, see [10, 41] or [42] for instances. The point of view defended in our proposal is guided by our context. It offers an unifying aspect between the different points of view of what an agent is. For us, an agent is emphasized by the following facets:

- Autonomy, in the sense of Wooldrigde [42], that is an agent exercises some kind of control of its own actions to make decisions without the guidance of others.
- Independence, in the sense of Franklin and Graesser [19], that is an agent acts in pursuit of its own agenda.
- Evolution, in the sense of Tokoro [41], that is an agent makes effort to maximize benefit to itself, and is in continuous evolution.

In such a context, 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. In a multiagent system, agents communicate together, and this capability is the key issue of the problem solving. 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.

In addition, the concept of agent is often compared with the one of concurrent object or actor. An actor is an active object, independent, seen as a concurrent task. Actor programming allows the achievement of concurrent programming without facing of hardware parallel feature and stresses the concurrent development of problem solving [11].

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 more than an actor, it is *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. An agent *interacts* in a concurrent and an asynchronous way with its environment to draw up a state of the other agents. It reasons to provide the most efficient behavior. Therefore, an autonomous agent should dispose of different abilities such as perception, action to keep control on all these activities. Actions are then controlled according to the state of the agent and environment. 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, 15, 21] or [33]. 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, 13, 16] or [38].

2.2. *Are Agents Adequate to Model Complex Systems in Geophysics?*

Many geophysical researchers have found out common features for natural phenomena: for instance, it is surprising how frequency, repetition or length strangely follow some rules. To understand these rules, the classical approach is to aggregate data, knowledge and hypotheses to build a model of the physical world. This model[2] is generally expressed by mathematical relationships between variables, matching some real physical magnitudes. Simulations are then used to analyze the properties of theoretical models. Results can then be processed and exploited with the help of statistical techniques to verify the given hypotheses.

However, 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 cannot avoid 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. Second, the expression of the complexity is poor. 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 during the simulation. Nevertheless, individual behaviors performed at a microscopic level will be responsible of the system evolution. Both system evolution and internal factors of the organization are not taken into account.

The study of such complex system in Geophysics has led to the concept of *Self-Organized Criticality* [2], to explain the "repeatability" of phenomena in nature. It has been proposed as a possibly general explanation for the presence in nature of satisfying conditions setting off phenomena. It describes the property of some systems to evolve naturally towards a stationary state where events take place. When such a state is hit, a small external perturbation could generate a large-scale phenomenon without predicting when it could appear. The best well-known example illustrating this property is the repeatability of avalanches in a sand heap: 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 cannot be reduced to those of its components. This non-reduction is assigned to the presence of interactions that dynamically unify the system components, and from which a phenomenon appears by affecting a part of the system.

With this approach, "classical" artificial intelligence techniques cannot provide enough semantics to give satisfying results. This relies on the fact that a centralized program remains too close to its reasoning, and is unaware of its environment. Tiny mechanisms cannot be deeply taken into account, leading to an inadequate description of the real world. Individual actions taking part of the elaboration, and therefore the organization of the system are being ignored.

The agent paradigm brings a new solution to the previous issues. It allows to decentralize the control activity by delegating agents to accomplish specific duties. From this observation, we propose an agent model[3] to represent the real world in autonomous parts, with interactions and evolution capabilities. Even if this point of view leads to a weak notion of what an agent is [43], 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 [31], 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 agents architecture 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 [29]:

- How to organize a multiagent system so that intelligent behaviors arise as a result of agent asynchronous interactions? Due to the underlying complexity of the system, we emphasize 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?

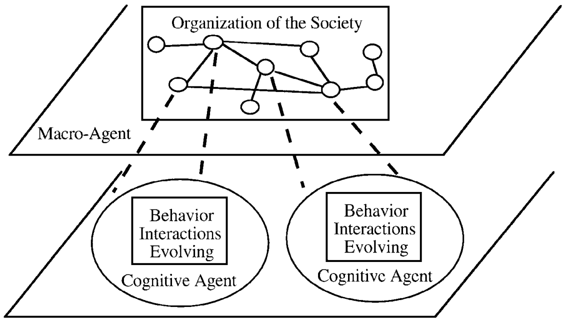

Figure 1. Traditional architecture of cognitive agents.

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, representing a cognitive architecture.

This type of architecture first identifies a macro-agent where the system behavior will be observed. The macro-agent organizes agents so that agents form a kind of society. Second, it describes an agent as a distributed and encapsulated element. The novelty of our approach is based on the following considerations: on the one hand, complex systems modeling needs to split the complexity into atomic pieces to better understand, control, and isolate underlying phenomena; on the other hand, agents cannot 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 of judiciously describing sophisticated phenomena. To increase simulation features and introduce a degree of complexity in our architecture, the GEAMAS approach is to marry cognitive and reactive architectures. Such an architecture is hybrid, according to the terminology found in [43].

In this architecture, the real world is arranged into three independent and asynchronous levels whose aim is to distribute complexity. Top layers deal with information at increasing abstract levels. The third level is the lowest layer where both knowledge and behavior are simple. Such agents are called reactive agents, to emphasize that actions are taken without consulting an internal model of reasoning. The second level meets intermediate goals and represents structures masking the

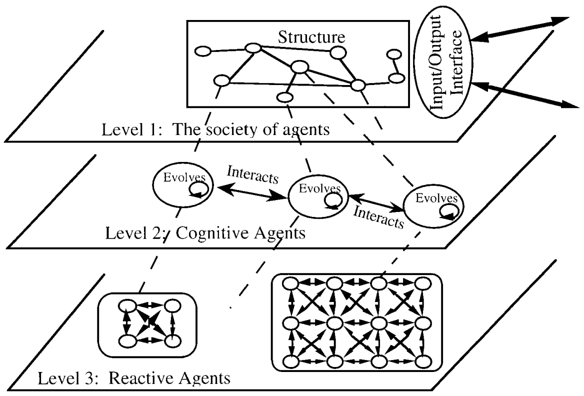

Figure 2. Three abstraction levels to model the complexity of a system.

complexity of the third level. It makes the link between the global goal of the whole system and reactive behaviors. Finally, the first level is the place where global behavior is observed and results collected. Figure 2 illustrates such an architecture.

The following sections detail the three levels of the GEAMAS framework.

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, interface and the whole organization of the system.

The role of the system is expressed through an external behavior, responsible of the simulation results. Such a behavior is driven by an input/output interface with the external world, for instance, a graph plotter or an Excel-like software tool. This interface is also intended to define and set input and output parameters for simulation needs.

The most important part of the first level is to describe how agents are organized to form a society. The organization of agents is represented through inter-agents connections, called acquaintance relationships in the actor model. This organization is defined as the description of agents' dynamic relationships used to structure asynchronous message within the system. It cannot be confused with a hierarchical tasks view leading to a functional division, as mentioned in Fox's excellent paper [18], to be illustrated in human

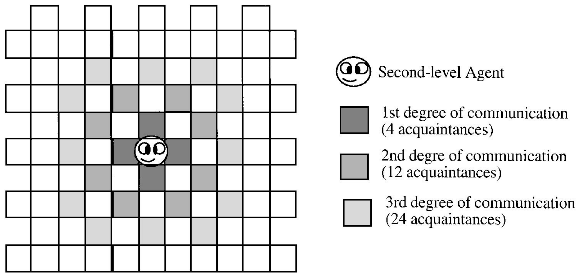
Second-level Agent
1st degree of communication (4 acquaintances)
2nd degree of communication (12 acquaintances)
3rd degree of communication (24 acquaintances)

Figure 3. Degree of communication scaling in GEAMAS.

organization or economics. This kind of hierarchical organization requires a tight coordination between agents. In natural phenomena modeling, it is quite different. No coordination can 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 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 of adaptability 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. It allows the user to design a real world 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 degree of communication describes the influence an agent can have on another. Furthermore, it 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.

However, modeling the 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 (3D grid), by using 3D 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. Figure 4 illustrates a third degree of communication.

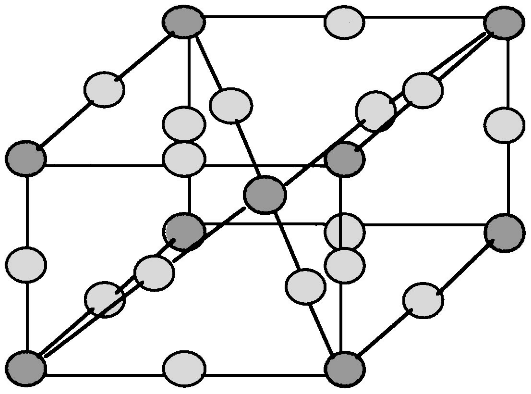

Figure 4. A 3D grid example of a communication network.

3.3. *The Second Level and the Agent Model*

The second level of the architecture describes the components of the society. Each one is seen as an autonomous agent describing high-level, *cognitive* agents. They will often be called *second-level agents* hereafter. A second-level 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 agents of the lowest level, more particularly the result of their interactions. Thus, a second-level agent meets intermediate goals by synthesizing more complex behaviors than those of the lowest level.

Each second-level agent is built from a model, called the *agent model*. The GEAMAS agent model describes 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 are the key issue of emerging behavior in the system. They 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 implements both the autonomous and independence parts of the agent. Agent's autonomy is described by a kind of control which determines room for decision making, as independence assumes that an agent can work without external solicitations. Nevertheless, internal behavior can be disturbed by some events that may occur during simulation. External behavior is the agent's response to such external solicitations.

- 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 nonlinear modifications as far as the simulation progresses. Modifications affect both the agent's data and its behavior, and depends on what was done by the agent during its life.

In GEAMAS, two kinds of dynamic evolution have been considered: structural modification through reorganization of the acquaintances network and "behavior emergence".

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 [35]: when a part of the system self-organizes, their components can be forced in their turn. The resulting constraint is playing a major role for the future behavior of the system. Let consider an avalanche in a sand heap. Self-organized structures will be created to model such avalanches. Retroaction is then used to specify that no energy remains in sand grains, and there is no avalanche 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 is very near of self-organized criticality model. 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 second type of dynamic evolution is the behavior emergence. It is a very interesting feature to investigate, as it contributes to the autonomous part of an agent. It authorizes an agent to reproduce adequate behaviors, i.e., behaviors the best adapted to the environment and the current situation. Second-level agents are in charge of this kind of modeling. All reactive agents pursue a common goal. By interpreting what takes place in reactive agents, cognitive agents get information concerning the underlying processes brought in action. The interpretation of such information thus governs the intelligent group behavior.

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, 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. We go further into this important feature of the model in Section 4.2.

3.4. Reactive Agents and Third Level

The third level of the architecture is composed of reactive agents describing second-level agent's components and their 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. Ferber demonstrates in [17] that the intensity of a signal emitted by an agent decreases as a function of the distance between agents, and that agent's behavior is then strongly dictated by their relative position in the topological structure. In GEAMAS, reactive agents evolve in a world whose structure is defined by the agents' communication network (2D grids, 3D 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 caused by the external event. This 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[4,5] 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. It also provides 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 movement and air molecules). Since the fourth stage, the structures are too small for further dissection. This characteristic is reproduced into the GEAMAS architecture: propagating agents dissection would make the system unrealistic. A micro-agent describes the most tiny grain and cannot be decomposed again without being out of the system's goals. The limit of precision is reached when the agent model cannot be inscribed in the expected grain level. As far as this limit can then be found, three abstraction levels are sufficient to distribute the complexity within the system.

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 [24]. As we have pointed out before, communicating by interactions is the essential condition to the emergence of the system global behavior. Each agent can communicate with every other, and in that sense, the system is fully connected [19]. Two possible communication paths are identified through the architecture: interactions between agents belonging to the same level (intra-level), and interactions between agents belonging to different levels (inter-levels). Intra-level interactions are performed through standard asynchronous message passing, according to the actor computing model. This is the point of view embodied within our architecture.

Two basic mechanisms are defined in the architecture to achieve inter-levels interactions: *Recomposition*, from micro-agents up to macro-agents and *Decomposition*, from macro-agents down to micro-agents. Such interactions are powerful, they expresses dynamic conceptual links. They allow information transfer between two different abstraction levels. Decomposition and Recomposition define two primitives of message passing with their own semantic. They are also performed through asynchronous messages, but are cut off from intra-level ones due to their specific semantic contribution.

A i-*level* agent can only recompose with a $i-1$-*level* agent if the $i-1$-*level* agent is the agent's 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 a high-level behavior to be implemented, thanks to the interpretation of information coming from lower levels computation. In the aggregation relationship, a component cannot be deleted without losing a part of its semantic; Decomposition and Recomposition work in an open environment, i.e., adding or deleting components does not change the agent's semantic. They just express some specific message types that circulate within the architecture at run time.

Ishida et al. [30] 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 needs to adapt itself to dynamically changing situations. In the authors' context, the system is a problem solver based on a production system. 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 levels in the system.

Moreover, in 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 the agents, and are only used to set off some specific mechanisms of the receiver. Our mechanisms are detailed now 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 in the system. It transfers information from level i to level $i-1$, for instance 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.

State parameters values 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. With a self-organized criticality point of view, we can understand why ice cube melts, in 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 micro-agents and combines them to be adapted to the situation. Recomposition then requires a high-level agent 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, micro-agents are assigned to molecules' melting, and we observe the high-level behavior as the ice cube to be shrunk. In cases, the dynamic of low-level agents governs emerging behaviors.

4.2.2. Decomposition. On the contrary, Decomposition allows information transfer from level i to level $i+1$. 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 has 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 that are the best able to process them. Decomposition then defines a kind of filtering process which is responsible for localizing the necessary agents. 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 that must be observed and shared 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. It is thus at the right place to express such knowledge; it may constrain a part of the system, either by acting on the world structure or on individual micro-agents. For instance, in the ice cube example, a global constraint may indicate how ice cube keeps its original shape when melting.

The next section gives some details about the implementation of the architecture and associated mechanisms, 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 [23], 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 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, 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 [22]. 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 behavior emergence out of micro-agents’ behaviors.

For our concerns, the GEAMAS architecture models several agents interacting with one another and embedded into three grain levels, where global behaviors dynamically emerge from the interpretation of micro-agents’ 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.

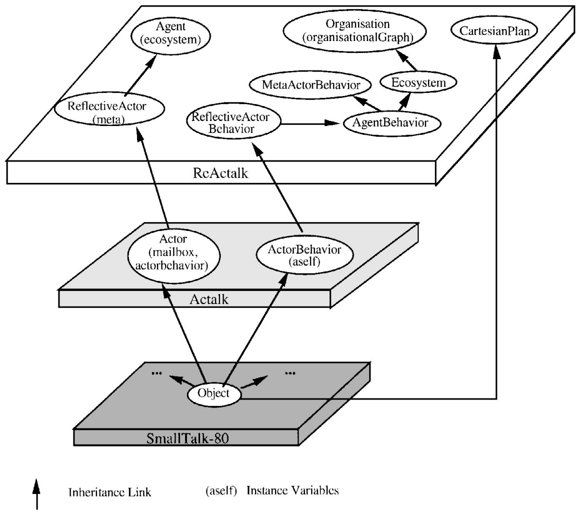

Figure 5. Simplified view of ReActalk, a multiagent development tool.

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.

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 describing 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 creation, an actor consults its mailbox permanently and answers potential messages sent to it. This messages examination 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 the Agents Society. Each agent moves in a global entity representing the society of agents, which forms the *Ecosystem*. Originally, the name of this class was chosen by analogy with ecological applications first derived from ReActalk. The Ecosystem class naturally inherits from the Agent Behavior 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.

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

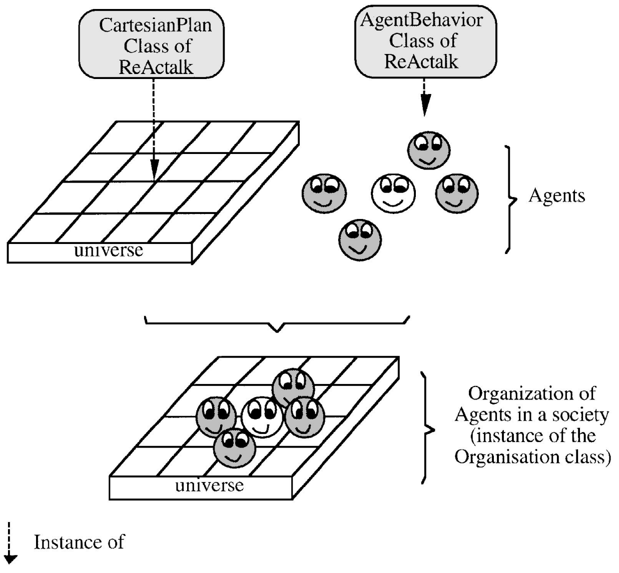
CartesianPlan Class of ReActalk
AgentBehavior Class of ReActalk
Agents
universe
Organization of Agents in a society (instance of the Organisation class)
universe
Instance of

Figure 6. Organization of agents in a CartesianPlan within ReActalk.

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 *MetaActor-Behavior* Class, which implements the way how behavior emerges, 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 also stored in the *meta* instance variable of the corresponding *ReflectiveActor*, 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. Figure 7 shows how the reflection mechanism is used in the GEAMAS context, when a cell, a second-level agent or a macro-agent is created.

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

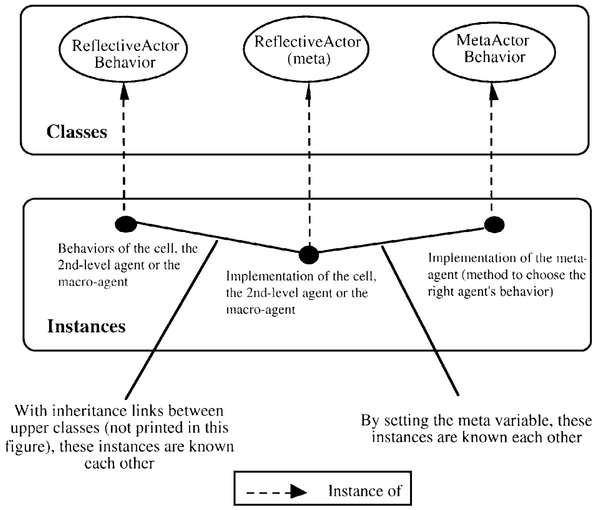
ReflectiveActor Behavior
ReflectiveActor (meta)
MetaActor Behavior
Classes
Behaviors of the cell, the 2nd-level agent or the macro-agent
Implementation of the cell, the 2nd-level agent or the macro-agent
Implementation of the meta-agent (method to choose the right agent's behavior)
Instances
With inheritance links between upper classes (not printed in this figure), these instances are known each other
By setting the meta variable, these instances are known each other
Instance of

Figure 7. Using meta-agents in GEAMAS.

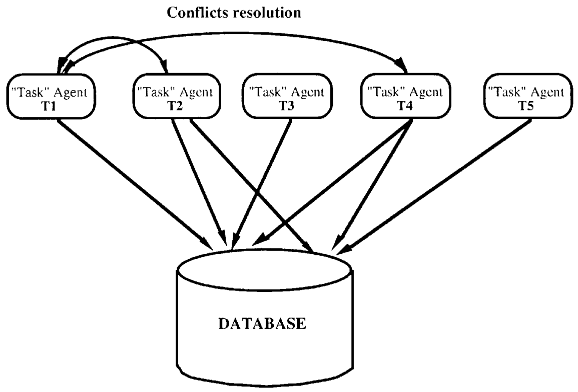

Figure 8. A bottom-up approach to build multiagent systems.

in agent-oriented design, before proposing some hints to help designers to build applications in the GEAMAS context.

6.1. *Specific Considerations of 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 is the case of "software agents" which live in computer systems, databases and networks. 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 [28]. In this case, the application is built by adding agents to the domain, where agents work 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 [29]. Due to existing constraints, this is the case in most industrial projects. The software design method is then *bottom-up* oriented, as shown in Fig. 8.

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 [40], 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. We here assume 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 of the agent, as well as external behaviors are modeled at the same time, and encapsulated together in agents. Our method follows this idea; it adopts an external perspective from which agent contents can be looked at. This relates to Gasser's ideas [20] 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 [12] or OMT [39], the first phase is to understand the real world, by looking at "business objects". This task can, moreover, be 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 to be implemented as agents, according to external specifications.

In GEAMAS, the designing task is quite well-structured, because the designer drives through a three-level architecture, and because a model of both agent and 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 [36]. 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 internal knowledge and behaviors, interaction processes, and evolution facilities of the agent. In addition, GEAMAS needs to look for the *taskability*, answering the following question: "at what level the activities and functions performed by an agent should be described?". This is a very important step of the designing task, as it determines the right level of complexity and granularity of the system. Furthermore, 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 underlined:

(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 into tiny pieces. Since a third-level agent cannot be decomposed again, it describes the most tiny grain. The right level of granularity is reached when the designer cannot describe in any more detail the agent, following the triad "interaction, behavior and evolution". If the model is not matched, a software component cannot be considered as agent, even in a reactive universe. At this step, also the stability of a reactive agent 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. For instance, when the temperature of water molecules falls down 0°C in the ice cube. State values will then be computed, according to messages handled by the cell during the simulation. This 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 micro-agents, the first-level agent models the resulting behaviors of the simulation, 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. But indirectly, they are expressed by the underlying structure of cells. 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.[6] 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 ny nonlinear processes and by small and slow perturbations. On such a basis, the volcano can be considered as a self-organized critical system [27]. This property has been revealed by numerous data surveys on Piton de la Fournaise, for

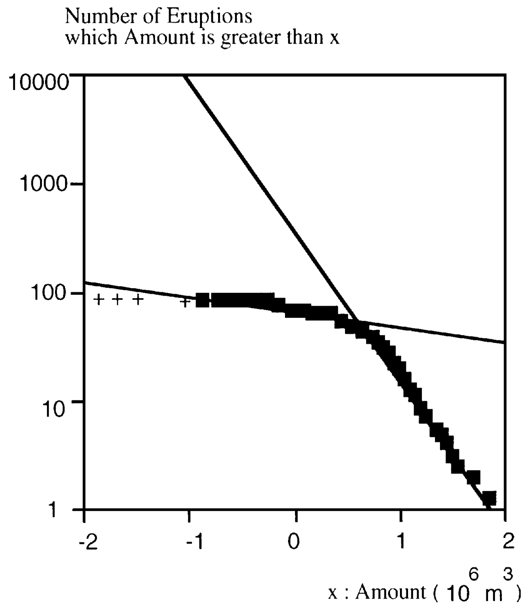

Figure 9. Number of eruptions according to size, occurring between 1920 and 1992.

instance, eruption size distribution over the last seventy years [26]. When focusing on eruption size distribution, data have then been noted as regular representations, illustrated by two characteristic segment lines (see Fig. 9).

The behavior of the volcano is completely determinist, because it follows a well-determined law, but at the same time, is driven by unpredicable 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 explore their relevance to the complexity of the eruptive behavior of the Piton de la Fournaise volcano. The issue was to establish if the complexity of natural phenomena emerges either from geometrical and material heterogeneity, or from the chaotic behavior related to underlying nonlinear equations. In such a context, studying local and distributed internal processes by simulation constitutes 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 interconnected 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 eruptions. 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. Micro-agents' behaviors encapsulating complexity were 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 limit given by a threshold. To recompute the internal pressure after draining, we need some physical laws. Such laws are called back pressure laws.

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:* When Gaussian, the back pressure law is expressed by a Gaussian law.

Lens' internal behavior describes how the 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 some rocks, a dyke appears. A dyke is modeled by several rock cells (with high values of the resistance), that gives the length. Thus, fluid exchanges depend on the geometric structure of the network. At the beginning of the simulation, resistance and pressure values are randomly chosen between two given limits.

First-Level Design. As regards with our immediate concerns, society behavior is 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 simulation 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 in each lens, continuous or discrete, for instance. 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 a new injection does not significantly affect internal parameters.

Output parameters fix the way to collect results. Two types of collection have been implemented in GEOMAS: counting the amount of ejected magma from all eruptions or counting one eruption and the amount of magma it ejects.

Some other parameters are also important: the degree of communication, determines the size and the topology of the network, as well as the number of lenses and rocks for a specific simulation. In addition, the network can be initially chosen either loaded or unloaded. When loaded, each initial lens pressure is triggered by a value which is just under 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 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 are given by reactive agents modeling the lens sides. When some parameter thresholds are reached, the cell alerts its society by a Recomposition message. Information hold in the message is then 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 Decomposition.

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 a 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 and be ready for the arrival of new disturbances. 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 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. 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, a 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 provokes an isolated eruption. In the example illustrated in Fig. 10, 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 to 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.

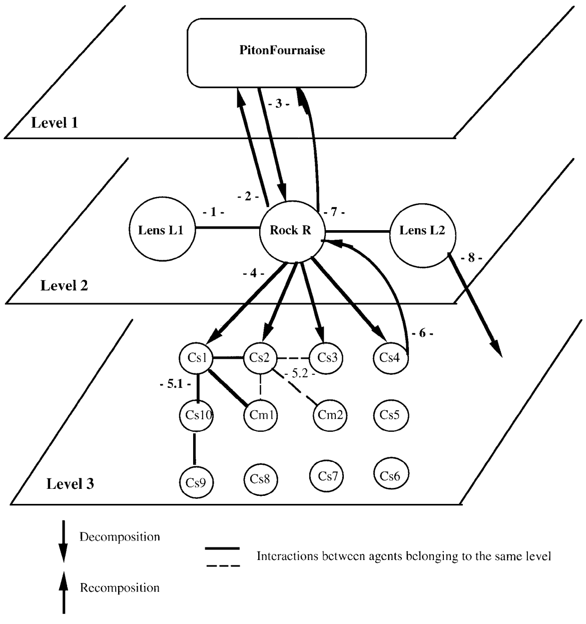

Figure 10. An easy example of how to work in GEOMAS.

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

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 cannot 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, 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 micro-level up to a macro-level:

- 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 macro-level down to a micro-level, 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., 50×50 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).

At the beginning we had a 2D conceptual model, where each agent communicated with its four closest spatial neighbors, but later this number was extended to

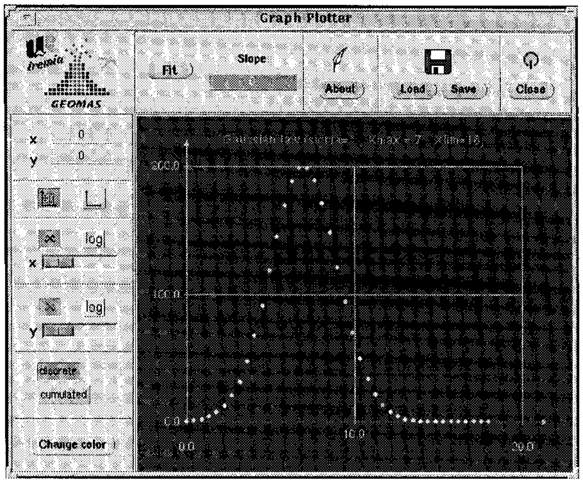

Figure 11. Simulation results provided by the GEOMAS graph plotter.

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 homogenous 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 cannot 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 interconnected, 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. With this experimentation, the GEAMAS architecture 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. It 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 as 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 [34, 35] 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 in artificial life and ecological systems.

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 [14]. 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 [37].

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 [25]. 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 helpful comments on various aspects of this work. This project is supported by the Institute of Research in Mathematics and Computer Science of the University of La Reunion (France).

Notes

1. Acronym for GEneric Architecture for MultiAgent Simulations.
2. The word "model" is here understood in a mathematical sense.
3. The word "model" is here synonym of agent model, in a computer science sense.

4. An agent of level i will be noted *i-level agent* in the rest of the paper.
5. In the rest of the paper, writing i assumed $i \in [1 \ldots 2]$.
6. Acronym for GEOphysics and MultiAgent Systems.

References

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,” *Physical Review*, vol. A38, pp. 364–374, 1988.
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 and 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*, edited by Bond and Gasser, Morgan Kaufmann Publishers: San Mateo, CA, pp. 3–36, 1988.
6. M.E. Bratman, D.J. Israel, and M.E. Pollack, “Plans and resource-bounded practical 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*, LNAI, vol. 890, Heidelberg, Germany, edited by M. Wooldridge and N.R. Jennings, Springer-Verlag, pp. 1–39, 1994.
10. C. Castelfranchi, “To be or not to be an ‘agent’,” in *Proc. European Conference on Artif. Intell., Workshop on Agent Theories, Architectures, and Languages, Intelligent Agents III*, LNAI, vol. 1193, edited by J.P. Müller, M. Wooldridge, and N.R. Jennings, Springer-Verlag, pp. 37–39, 1997.
11. F. Cicasele and V. Loia, “A fuzzy evolutionary approach to the classification problem,” *Int. Journal of Intelligent and Fuzzy Systems*, 1998, in press.
12. P. Coad and E. Yourdon, *Object-Oriented Analysis, Object-Oriented Design, Yourdon Press Computing Series*, Prentice Hall: Englewood Cliffs, NJ, 1991.
13. Y. Demazeau and J.P. Müller, “From reactive to intentional agents,” in *Proc. Decentralized Artificial Intelligence*, vol. 2, edited by Y. Demazeau and J.P. Müller, Elsevier North-Holland, pp. 3–14, 1991.
14. A. Drogoul and J. Ferber, “Multiagent simulation as a tool for studying emergent processes in societies,” in *Proc. Simulating Societies: The Computer Simulation of Social Phenomena*, edited by N. Gilbert and J. Doran, North-Holland, 1994.
15. E.H. Durfee, V.R. Lesser, and D.D. Corkill, “Coherent cooperation among communicating problem solvers,” in *Proc. Readings in Distributed Artificial Intelligence*, edited by Bond and Gasser, Morgan Kaufmann Publishers: San Mateo, CA, pp. 268–284, 1988.
16. J. Ferber, “Reactive distributed artificial intelligence: Principles and applications,” in *Proc. Fundamentals of Distributed Artificial Intelligence*, edited by G. O’Hare and N. Jennings, Wiley and Sons, 1994.
17. J. Ferber, “Cooperation strategies in collective intelligence,” in *Proc. of the 7th Workshop on Modelling Autonomous Agents in a MultiAgent World*, Eindoven, The Netherlands, January 22–25, 1996.
18. M.S. Fox, “An organizational view of distributed systems,” in *Proc. Readings in Distributed Artificial Intelligence*, edited by Bond and Gasser, Morgan Kaufmann Publishers: San Mateo, CA, pp. 140–150, 1988.
19. S. Franklin and A. Graesser, “Is it an agent or just a program?: A taxonomy for autonomous agents,” in *Proc. European Conference on Artif. Intell., Workshop on Agent Theories, Architectures, and Languages, Intelligent Agents III*, LNAI, vol. 1193, edited by J.P. Müller, M. Wooldridge, and N.R. Jennings, Springer-Verlag, pp. 21–35, 1997.
20. L. Gasser, “Conceptual modeling in distributed artificial intelligence,” *Journal of the Japanese Society for Artificial Intelligence*, vols. 5–4, July 1990.
21. L. Gasser and M.N. Huhns, “An analysis of research in DAI,” in *Proc. Readings in Distributed Artificial Intelligence*, edited by Bond and Gasser, vol. 2, Morgan Kaufmann Publishers: San Mateo, CA, pp. 3–35, 1988.
22. S. Giroux, “Agent et acteurs: Une nécessaire unité,” Ph.D. thesis, University of Montreal, Canada, March 1993.
23. S. Giroux, “Open reflective agents,” in *Intelligent Agents vol. II: Agent Theories, Architectures, and Languages*, LNAI, vol. 1037, Montreal, Canada, edited by M. Wooldridge, J.P. Müller, and M. Tambe, Springer-Verlag, pp. 315–330, 1996.
24. 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*, PAAM’96, London, UK, April 22–24, 1996, pp. 295–310.
25. 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.
26. 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.
27. 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.
28. B. Hayes-Roth, “A blackboard architecture for control,” *Artificial Intelligence*, vol. 26, pp. 251–321, 1985.
29. M.N. Huhns, Foreword of “Multiagent systems: A theorical framework for intentions, know-how, and communications,” LNAI, vol. 799, by M.P. Singh, Springer-Verlag, 1994, pp. I–XII.
30. T. Ishida, L. Gasser, and M. Yokoo, “Organization self-design of distributed production systems,” *IEEE Transactions on Knowledge and Data Engineering*, vols. 4–2, pp. 123–134, 1992.
31. N.R. Jennings, “Specification and implementation of a belief desire joint-intention architecture for collaborative solving

problems," *Journal of Intelligent and Cooperative Information Systems*, vols. 2–3, pp. 289–318, 1993.

32. C.B. Langton, "Computation at the edge of chaos: Phase transition and emergent computation," *Physica D*, vol. 42, pp. 12–37, 1990.
33. 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.
34. 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*, vol. 1060, Montreal, Canada, Springer-Verlag, June 1996, pp. 75–83.
35. P. Marcenac, "Modeling multiagent systems as self-organized critical systems," *31th Hawaii International Conference on System Sciences*, HICSS-31, IEEE Computer Society Press, 1998.
36. P. Marcenac, S. Giroux, and J.R. Grasso, "Designing and implementing complex systems with agents," edited by I. Jelly, I. Gorton, and P. Croll, Chapman & Hall, pp. 27–38, March 1996.
37. P. Marcenac, S. Leman, and S. Giroux, "Cooperation and conflicts resolution in multiagent systems," in *Proc. 34th ACM SouthEast Conference*, Tuskegee, AL, edited by K.H. Chang and J.H. Cross, pp. 289–291, April 17–19, 1996.
38. 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*, LNAI, vol. 890, Heidelberg, Germany, edited by M. Wooldridge and N.R. Jennings, Springer-Verlag, pp. 261–276, 1994.
39. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall: London, England, 1991.
40. 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.
41. M. Tokoro, "An agent is an individual that has conciousness," in *Proc. European Conference on Artif. Intell.*, *Workshop on Agent Theories*, *Architectures, and Languages*, *Intelligent Agents III*, LNAI, vol. 1193, edited by J.P. Müller, M. Wooldridge, and N.R. Jennings, Springer-Verlag, pp. 45–46, 1997.
42. M. Wooldridge, "Agents as a Rorschach test: A response to Franklin and Graesser," in *Proc. European Conference on Artif. Intell.*, *Workshop on Agent Theories*, *Architectures*, *and Languages*, *Intelligent Agents III*, LNAI, vol. 1193, edited by J.P. Müller, M. Wooldridge, and N.R. Jennings, Springer-Verlag, pp. 47–48, 1997.
43. 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*, LNAI, vol. 890, Heidelberg, Germany, edited by M. Wooldridge and N.R. Jennings, Springer-Verlag, pp. 1–39, 1994.