Ontic Computing: A New Paradigm for Software Agent

Eric Sanchis

Laboratoire de Gestion et Cognition
IUT Ponsan - Université Paul Sabatier
115 route de Narbonne,
31077 Toulouse, France
sanchis@iut-rodez.fr

Abstract. Stemming from multiagent systems and systemics, the aim of ontic computing is the study of the complex properties of software agents. For this purpose, it provides a unified conceptual framework called *Systemion Model* built on the concepts of external finality, internal finality, qualities and attributes.

1 Introduction

In the image of the companies of today, the new computing systems to be designed are increasingly complex (*autonomic systems* [6], *grid computing* [4]. A major challenge for the engineers is then to understand the complexity of such entities. A manner of approaching this complexity is to consider these systems as a set of interacting elements, provided with different properties of variable complexity. Designing such systems then consists in identifying the relevant elements and after that to understand and control their interactions.

In computing, two communities of scientists study this type of systems: researchers in *multiagent systems* (*MAS*) and researchers in *distributed systems*. Although interested in different problems, the first ones conceive their applications in terms of agents provided with abstract properties and sophisticated interactions such as co-operation, coordination and high level communication. The second ones reason in terms of processes or threads, i.e. by means of low level active entities, and mechanisms of interaction perfectly controlled like signalization, synchronization and communication. In other words, the first ones regard the agents as high level entities, the second ones as a technique of sophisticated programming.

Today, more and more researchers consider that these two fields have common research topics - even if their problems remain distinct -, and that these two disciplines can make each other progress [3], [9]. The generally advanced argument to promote this coming together is that MAS deal with many problems where the distribution of the agents plays a central role and distributed systems constitute a conceptual and operational platform which is ideal to develop them.

However, the association of these two fields of research remains insufficient when it is a question of treating what constitutes one of the specificities today's computing systems: their complexity. The weakness of the MAS approach, weakness also shared by the distributed systems approach, is that they integrate neither in their models, nor in their practices, *a conceptual framework* adapted to the study of *complex systems*.

In order to fill this gap, we introduce a particular systemic line of thought called *systemics of complexity*, which we associate to the two preceding approaches. It results a unified model called the *systemion model*. The matter of this paper is to present the fundamental theoretical choices which constitute the core of *ontic computing*, point of convergence of distributed systems, MAS and systemics (Fig. 1).

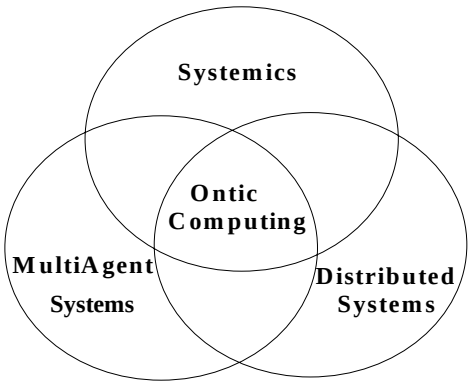

Fig. 1. Ontic Computing

This paper is structured as follows. Section 2 introduces complex systems by putting them in opposition with what one generally calls the complicated systems. Then an abstract model resulting from systemics and adapted to the study of the complex systems is presented. In the following section, the problem of complexity is circumscribed with the field of the software agents. Lastly, part 4 presents within a single framework called *systemion model* an articulation of the various concepts brought out in the preceding sections, and then applies this model to complex software agents.

2 Complex systems

Various causes seem to be at the origin of the complexity of a phenomenon: the number of elements, the diversity and the nature of the interactions between these elements, the mixture of determinism and non determinism, order and disorder, cascading or circular internal transformations, the presence of emergent processes.

2.1 Complicated systems and complexity

It is frequent to confuse *complicated system* and *complex system*. This confusion is often caused by the same great number of elements which compose the two types of systems and by the difficulty of understanding the interactions occurring between their elements. Moreover, in both cases the *system* term means *representation* or *pre-representation* of the studied phenomenon. However, there is a difference in nature between these two categories of systems: a *complicated system* can skilfully be simplified in order to be able to carry out a particular calculation, whereas a *complex system* cannot be reduced by some handling to a *complicated system* under penalty of losing the intelligibility of the studied phenomenon.

To study a complex system, an observer can use two types of approaches: *equational* approach and *structural* approach. The choice of one or the other will be primarily guided by the objective to reach: to calculate or to understand. The characteristics of these two approaches can be stated in the following way.

Equational approach:
- It does not take into account the internal composition of the system elements
- The internal composition of an element is not regarded as a source of complexity of the system: the constitution of an element (internal architecture, components) is transparent for the observer
- The elements are many and indistinguishable: they are considered as *particles*
- It uses powerful mathematical formalisms
- It favours obtaining quantitative results.

Structural approach:
- It takes into account the internal composition of the system elements
- The internal architecture of the element is regarded as a source of complexity of the system
- It puts up with the presence of heterogeneous elements
- It proceeds by analysis and synthesis
- It supports obtaining qualitative results.

These two types of approaches are not antagonistic but serve objectives of different nature. To give clear ideas, let us give some examples of fields and tools used by these two approaches. The equational approach is employed in economy, biology or meteorology. The tools used are primarily mathematical ones (nonlinear differential equations). The structural approach is at the heart of the sciences of organizations and a broader way of the social sciences where systemic modelling is moulded perfectly with the needs of researchers. Our reasoning consisted in adapting the systemic approach of complex systems to the field of software agents.

2.2 Systemics of complex systems

The object of Systemics is the qualitative study of complex systems, systems being able to be of unspecified nature (physical, biological or social). Contrary to the traditional methods known as Cartesian or analytical, it does not aim at predicting (to calculate) the evolution of a complex system, but rather making it understandable. Remote heir of Cybernetics [12] whose object was to make the action effective, Systemics rather aims at understanding the nature of the action.

Scientific theory for some, intellectual crutch for others, Systemics offers with its users a set of conceptual tools as abstract as delicate to handle. The systemic approach is often perceived as difficult even diverting for a *modelisor* trained with the traditional methods of analysis of the systems. Whereas he would break up a complex system in order to extract from it the essential variables which will enable him to carry out a particular calculation, the systemician would also break up this same complex system into simpler elements but without reducing the comprehension of the whole to those of its parts, i.e. by preserving its total intelligibility (property of *near decomposability* of the studied system [11]).

One of the consequences of the systemic paradigm is that the systemician proposes *a possible model* of the complex system and not a final mathematical function. This possible model reflects the project of the modelisor and specifies at the same time the limits of the project and of the model. As it is a possible, there is not an isomorphism between a complex system and a model supposed to represent it: several complementary models can coexist in the representation (or construction) of a complex system.

The systemician Jean-Louis Moigne [7] worked out a generic model called *general system* (Fig. 2) adapted to the study of complex systems. This model includes four dimensions (Function, Environment, Finality and Transformation) and must be regarded as a mould thanks to which the observer/modelisor will represent the complex phenomenon by successive instantiations of these four dimensions.

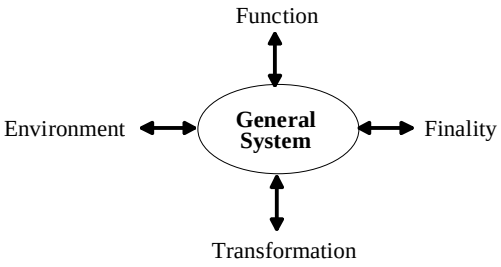

Fig. 2. General System

We adapted this model to the context of software agent in order to be able to study and establish complex properties of agents, some already explored in the field of MAS (e.g. *autonomy*), others much less studied (e.g. *finality*).

3 Complexity of the agents

Compared to the former work carried out in Artificial Intelligence, MAS introduced two major concepts: the agent and the interaction. Studying complexity within MAS consists in determining its demonstrations into the internal functioning of the agents and their reciprocal relations.

The interactions between multiple elements are a source of complexity known since the first work carried out on general systems (physical, biological, social systems, etc). This dimension of the complexity of a system is intensively studied in the equational approach. Our research [9] led us to identify a second source of complexity: *properties* of the agents. Indeed, the agents are often provided with many properties, very different from each other, with no commonplace mutual relationships and of variable complexity/complication. It is this second aspect of the complexity carried by the properties which we will present.

Several researchers [5], [1], [2] synthesized in the form of lists the properties most often mentioned. These lists broadly match; this is why we will present only one of them - that was worked out by Etzioni [2] - at the same time dense and homogeneous: an agent is able to take initiatives and to exercise a level of non commonplace control on its own actions (*autonomous*). It accepts high level requests stating what the user wants and is responsible for *how* and for *or* to satisfy the request (*goal directed*). The agent is able to engage in complex conversations with the other agents, individuals included, so as to obtain information or their assistance in the achievement of its goals (*co-operative*). The agent automatically particularizes itself with the preferences of its user, based on the former experience. It adapts also automatically to the changes of its environment (*adaptive*). An agent does not obey blindly the orders but has the capacity to modify the requests, to require clarifications or even to refuse to satisfy certain requests. The actions of an agent "are not hardwired"; it is able to choose the actions to be carried out dynamically and in the required sequence, in response to the state of its external environment (*flexible*). Contrary to standard programs which are directly launched by the user, an agent can see into the changes of its environment and decide when to act (*proactive*). An agent is in permanent execution (*temporal continuity*). It has a well defined personality and an emotional state. An agent is able to move from one machine to the other and through different systems architectures or platforms *(mobile*).

3.1 Complex properties

Properties of the agents, *qualities* and *attributes*:
When one wishes to integrate within the same agent two properties such as *mobility* and autonomy, it quickly comes to the conclusion that these two properties cannot be treated in a uniform way.

Our work led us to distinguish two categories of properties of deeply different nature: *qualities* and *attributes*. *Qualities* characterize properties with vague and elastic contours, with multi-dimensional models, with not easily measurable aspects (even non measurable) of an agent. Some properties which can be identified as qualities are *autonomy*, *sociability* and *intelligence*. *Mobility*, *replication* and *perception* are properties intrinsically less difficult to delimit than qualities. They are generally reduced to a mechanism, to one or some well defined modalities of functioning. We will call these properties the *attributes*.

Qualities, points of complexity of the agents:
To the more or less large simplicity of the attributes, we oppose the complexity of qualities. Whereas an attribute can be complicated to represent, to model or to establish, a quality is complex by nature, i.e. difficult (perhaps impossible) to have an intimate knowledge of it. Qualities are properties with badly determined borders, with the contents changing according to the points of view which one can have. They authorize multiple models and different interpretations, when these interpretations are not contradictory. Part of our work consisted in clarifying some points of divergence between the researchers who are interested in autonomy of artificial agents and synthesizing them in the form of criteria of analysis. These criteria include the problem of "metric of a quality", i.e. the problem which consists in determining if a quality is *measurable or not measurable*, and in the first case to identify the various relevant levels (or measurements). If the central quality of our research were the autonomy of software agents, the same questions and methods of study could apply to the intelligence of the agent.

3.2 Systemic agents

The study of the properties present in software agents showed that strong relations could exist between certain properties (e.g. mobility in relation to perception of sites, autonomy with mobility) and that the organization of these properties within the agent was not neutral with respect to the total behaviour of the agent. In order to specify the organization that we kept, we introduce the intermediate concept of *systemic software agent*. The purpose of this concept is to clarify the articulations between the properties kept and the total organization. It will especially enable us to derive various agent classes, such as *object-agents* or *ontic-agents*, which are at the center of ontic computing.

We define *a systemic agent* as a software entity
(a) located in an active environment, where there are reciprocal action and reaction between this environment and the agent,
(b) provided with a simple or complex finality,
(c) prone to internal transformations which allow the entity to evolve with time,
(d) provided with a software architecture allowing it to be executed into one or more systems.

This generic framework of software agents can be combined using various manners, according to the selected characteristics of the general definition mentioned above. Let us give two simple examples of agent classes derived from the systemic agent abstraction: when a designer of MAS considers that the single finality of the active entities of the system is the task which is associated to each one of them, this leads to the class of *object-agents* (Fig. 3), when a researcher concentrates on the software modifications which are internal to an agent and that these modifications are in connection with the action of the environment, the entity is included in the category of *open software agents* [8]. The object of the work presented in this paper relates to the class of the *ontic-agents*. They maintain a particular relationship with the finality, one of the dimensions of systemic agents.

4 The Ontic Paradigm

Philosophical subject for more than two millennia, finality has not ceased being dissected by *new sciences* such as physics, biology, psychology or systemics. Through centuries, finality was interpreted in various manners. Nevertheless, it is often regarded as the property for an action or an entity of tending towards a goal, to be organized according to an objective. Several types of finalities were distinguished by philosophers: the *magic finality*, *the intentional finality*, *the natural finality* or *non-finality* (negation of its existence). In front of a behaviour or an organism, finality is a matter of impression, feeling: finality leaves no palpable traces, it is not scientifically observable. Thus, finality has many common points with *autonomy*: hard to define it precisely, multiple or even contradictory interpretations, property more postulated than demonstrated. In our categorization of the properties of software agents, finality seems closer to a *quality* than to an *attribute*.

4.1 Finalities

In the field of artificial systems, the problem of finality is much simpler. Indeed, any computing system is de facto finalized, i.e. has an end. It is provided with an *external finality*, one of the components of the *natural finality*. In this context, the external finality materializes the utility of the system for a third party.

From a technical point of view, the external finality of a software agent is materialized in the form of a task to be achieved or a service to be provided: for example, a mobile agent must seek and recover information of a certain nature among a set of connected sites. In a more general way, agents with external finality are present in traditional or distributed applications, but are also studied in MAS.

A second component of the natural finality is the *internal finality*. Contrary to the preceding one, its aim is the software agent itself. The object of internal finality can be the maintenance of the internal integrity of the software agent when it is subjected to the pressures of its environment but also to optimize the use of the resources so

that the agent can function suitably. Two examples of artificial agents provided with an internal finality are *animats* [13] and *worms* [10]. The latter ensure their survival by replication on any host system which is accessible to them.

Lastly, software *viruses* are today the agents where it is possible to distinguish a combination of the two finalities. The external finality of this type of agents is materialized by the infection or destruction machinery while the internal finality is revealed in the capacity to perceive new systems where to reproduce. The concept of ontic agent aims at integrating the external finality and the internal finality within a single formal framework in order to build new types of agents with more useful functionalities.

Ontic agents:
We call *software ontic agent* (or *ontic agent*) a software agent which integrates the two aspects of *the natural finality*, i.e. *the external finality* and *the internal finality*. An ontic agent is provided with a property of existence independent from its functional capacities. They enable it to carry out a task, a work, to have a function in its environment. The property of existence enables it *to survive* in its environment independently of an unspecified assignment of task. The concretization of this property of existence can take several forms: survival by replication, mobility, combination of these various forms or others. Lastly, in an ontic agent the existence prevails over the function. Consequently, ontic agents bear some relation to time and to utility which is different from the other software agents.

In other words, designing an ontic agent consists in forgetting the possible utility of this agent to concentrate on the existential aspect of this one, on the way of making it last in an open and therefore unpredictable environment. It contributes to clearly dissociate what the agent *must carry out* (external finality) from what *materializes its "proper essence"* (internal finality).

Finality and software agent hierarchy:
The introduction of the finality makes it possible to identify three broad classes of active entities: *processes* (or *threads*) implemented in the distributed applications, the *object-agents* studied in the MAS, and the *ontic-agents* at the center of *ontic computing* (Fig. 3).

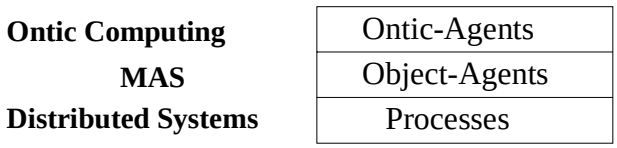

Fig. 3. Range of active software entities

Each one of these three classes defines a step where the finality of the active entities materializes itself differently. In the first layer, processes establish a commonplace external finality, that of the program which they carry out. At the higher level, the

active entities called *object-agents* are provided with a finality which will be also described as external, because directed towards the resolution of a problem the origin of which is external to the agent. However, this finality is more elaborate than the previous one because the agent can be induced to change its task in its cycle of life. For example, the agents studied within the framework of MAS are typically object-agents, i.e. objects of the problem for which they were designed. But from the point of view of the finality, one or more reassignments of tasks can take place within a MAS. A break-point is also present between the layers of object-agents and of ontic-agents: the latter also have an internal finality which, in a way, obliges them to calculate for their own needs too.

4.2 Systemion

The *Systemion Model* (*SYSTEMIc daemON*) [9] is a software architecture which articulates into a unique framework the concepts of *internal finality* and *external finality*, *qualities*, *attributes* and *tasks*. We call *systemion* a software agent built according to *the systemion model*. This software architecture can be split up into two subsystems:
(1) a *functional subsystem* which materializes the external finality, i.e. properties related to the fulfilment of the task possibly assigned to the agent. This task can change during the systemion life-cycle and constitutes the most flexible part of the systemion.
(2) a *behavioural subsystem*: it contains the properties (qualities and attributes) specific to the agent, that is the properties which are independent from the task which it will have to execute. One property plays a particular role in the systemion model: it is the *modification* attribute [8]. By its presence (or absence), the systemion will be able (or will not be able) to change its current task. This software layer implements the internal finality (Fig. 4).

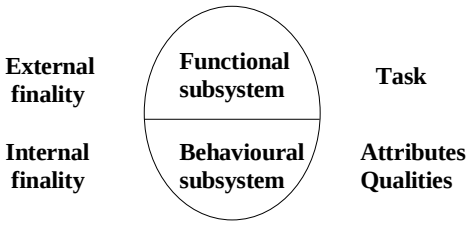

Fig. 4. Systemion Architecture

We can note that a conflict may occur between the two finalities. For example, the execution of a task leads a mobile systemion to move on a site previously reached and whose behavioural subsystem keeps the trace of a material or software incompatibility: the functional subsystem invites the systemion to move while the behavioural subsystem prohibited this displacement. The absence of connection between the two

finalities poses an internal problem of arbitration to which the model brings a proper answer: the behavioural subsystem takes precedence over the functional subsystem.

The *systemion model* and complex agents:
The *systemion model* is a particular model of systemic agent which proposes a specific interpretation of the concept of finality, a property present in many systems (natural or artificial) considered as complex. This generic model of agent was used to study what we established as being points of complexity of the agents, i.e. qualities. One of them was intensively the object of our work: autonomy. After establishing several criteria allowing to distinguish the different points of view which emerged from MAS literature about this quality, we elaborated a model of autonomy which precisely characterizes the link that this property can share with non-determinism (model of *autonomy with regard to an attribute* [8]). Non determinism is indeed one of the significant aspects of complex systems.

We indicated in section 3 that the interactions were also a source of complexity for the systems in which they occur. One of the features of complex systems is that they are physically opened to their environment. These *open systems* are characterized by a continual exchange of materials and energy, at the same time between the interior and the outside of them, but also between their internal components. For example, these dynamics allow a living organism to grow, develop, reproduce and to survive in spite of changing external conditions, while causing regeneration, renewal, destruction and replacement of the entities which compose it. The systemion model was used to analyze the impact of this physical opening according to whether it occurred on the level of the functional subsystem or of the behavioural subsystem of an agent [8]. As for non-determinism previously mentioned, the objective was to isolate one and only one facet of complex systems.

5 Conclusion

More than complicated systems, complex systems require a suitable reasoning. By breaking up element by element the complexity of a system then by injecting into a perfectly controlled software architecture – the *systemion model* - only one of these elements at the same time (a particular quality, non determinism, the physical opening, etc), it is possible to better study, aspect by aspect, the various faces of this complexity.

References

1. J. M. Bradshaw, "An Introduction to Software Agents", in "Software Agents", J. M. Bradshaw (ed), MIT Press, Cambridge (MA), USA, 1997.
2. O. Etzioni, D. S.Weld , “Intelligent agents on the Internet: Fact, Fiction, and Forecast”, IEEE Expert 10(4), pp 44-49,1995

3. J. Ferber, "Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence", Addison-Wesley, 1999.
4. I. Foster, "The Grid: A New Infrastructure for 21st Century Science", Physics Today, February 2002.
5. S. Franklin, and A. Graesser, "Is it an Agent, or just a Program ?: A Taxonomy for Autonomous Agents", in *Intelligent Agents III – Proceedings of the Third International Workshop on Agent Theories,Architectures and Languages (ATAL 96)*, (Edited by Müller J. P., Wooldridge M. and Jennings N.), Lecture Notes in Artificial Intelligence, 1193, Springer Verlag, 1996.
6. J. O. Kephart, D. M. Chess, "The Vision of Autonomic Computing", IEEE Computer, Vol. 36, no 1, 41-50, 2003.
7. J.-L. Le Moigne, "Formalisms of systemic modelling", http://www.mcxapc.org/docs/ateliers/0505formalismes.pdf
8. E. Sanchis, "A Systemic Framework for Open Software Agents", *Second International Workshop on Radical Agent Concepts* (WRAC 2005), 20-22 September 2005, Greenbelt (Maryland), USA.
9. E. Sanchis, "Designing new Agent Based Applications Architectures with the AGP Methodology" in *–Proceedings of the twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (*WETICE* 2003), IEEE Computer Society, 2003.
10. J. F. Shoch, J. A. Hupp, "The "Worm" Programs – Early Experience with a Distributed Computation", Communication of the ACM, Vol. 25, no 3, 172-180, 1982.
11. H. A. Simon, "The Sciences of the Artificial", MIT Press, Cambridge (Massachusetts), 1996.
12. N. Wiener, "Cybernetics, or control and communication in the animal and in the machine", The MIT Press, Cambridge (Mass.), 1947.
13. S.W. Wilson, "The Animat Path to AI", in J.A. Meyer and S.W. Wilson (Eds.) *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, The MIT Press, 1991.