A Systemic Framework for Open Software Agents

Eric Sanchis

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

Abstract. The systemic theory associates open systems and complexity closely. This article presents a particular articulation between these two concepts using the Systemion Model. Two types of opening are defined, characterized and illustrated using examples.

1 Introduction

Computer systems are more and more complex both at the level of their composition and of their behaviour. The complexity of these systems has become so extensive that the most experienced system administrators feel more and more difficulty in ensuring their good functioning. But how can a complex system be identified? According to our point of view, a *complex system* is composed of heterogeneous elements with not clearly defined components and fuzzy interactions, whose total functioning cannot be deducted from the functioning of its elements.

It is important to note that the computer systems do not constitute the only field where complexity poses a major intellectual challenge: complexity is also present in biological, physical and socio-economic systems. For half a century this inevitable complexity has given rise to numerous studies and researches which has led to two very different approaches: (1) equational approach and (2) structural approach.

The *equational approach* is articulated around an extreme simplification of the basic elements of the system, allowing the use of powerful mathematical formalisms the treatment of which has led to an interpretation of the total behaviour of the system [16], [1].

The *structural approach* refutes the reduction of the elements of the system in simple indiscernible items and postulates that it is possible to decompose the system into elements and interactions without denaturing their intrinsic complexity. Contrary to the equational approach, the structural approach does not suppress the internal architecture of the elements of the system. *Multiagent systems*, *autonomic computing* [10] and the various theories resulting from the systemic thought such as the *hierarchical complex systems* [15], the *systemography* [12] or *systemions* [14] belong to the structural approach. Besides, the structural method insists on the architectural charac-

teristics of the elements which compose the complex system. According to the relations which it maintains with its environment, the system will be called *open system* or *closed system*. We shall present two aspects completely different from the opening of a system: the *informational opening* and the *physical opening*. It is this last type of opening which we shall illustrate in a more precise way by applying it to the concept of agent. We shall use it to introduce its internal architecture (two-layer architecture) and identify the properties which introduce complexity within the agent (qualities). Linking the physical opening to an architecture with two levels (one layer ensuring the stability of the entity, the other taking care of its internal transformations) seems to be an interesting way of research in the implementation of self-management within complex computer systems.

This paper is structured as follows. Section 2 presents two definitions of the notion of open system, illustrating two different points of view on this concept. The first definition was proposed by Carl Hewitt in order to have a formal model of calculation adapted to the study of distributed systems. The second definition arises from General System Theory and gives itself a vaster field of applications - natural or artificial physical systems to the most complex social organizations. The interest of these two definitions is that each one of them introduces an aspect which is different from the concept of opening. The first is mainly based on the concept of informational opening; the second integrates the exchanges of materials between the system and its environment, i.e. what we will name the physical opening. The section will end with a very general description of a software agent model whose architecture allows the implementation of the physical opening. This model of agent results from one of the components of General System Theory: the systemic approach. The following two sections will introduce the supplementary abstract tools necessary for a precise presentation of an open agent model called the Systemion Model. Finally, the last section will illustrate the realization of the physical opening in this model by using two examples.

2 From Open Systems to Open Agents

The expression *open system* was used in various disciplines, in old ones (physics, biology) as well as new ones (sciences of the systems, data processing). Each one of them has provided this concept of characteristics, for some of them, very general and largely applicable to many contexts, for others very specific to a particular field. Three uses of this expression can be mentioned (1) it is sometimes used to characterize an elementary entity taking part in the more general activity of an organization, (2) in certain contexts, the expression even applies to the organization itself and not to its components, (3) in other fields, the expression is applied at the same time to the component and to the whole. The object of our work relates more to the architecture and the functionalities of the unit of execution than the characteristics of the organization to which it belongs. Consequently, after having specified what the concept of open system means, our matter will concentrate on the *open agent* concept.

2.1 Open Systems

Carl Hewitt defines and characterizes the open systems in the following way: "*Open systems deal with large quantities of diverse information and exploit massive concurrency. They can be characterized by the following fundamental characteristics:*

(1) ***Concurrency****. Open systems are composed of numerous components such as workstations, databases, and networks. To handle the simultaneous influx of information from many outside sources, theses components must process information concurrently*

(2) ***Asynchrony****. There are two sources of asynchrony in open systems. First since the behavior of the environment is not necessarily predictable by the system itself, new information may enter the system at any time, requiring it to operate asynchronously with the outside world. Second, the components are physically separated distances prohibiting them from acting synchronously. Any attempt to clock all the components synchronously would result in an enormous performance degradation because the clocks would have to be slowed down by orders of magnitude in order to maintain synchronization*

(3) ***Decentralized control****. In an open system, a centralized decision maker would become a serious bottleneck. Furthermore, because of communications asynchrony and unreliability, a controlling agent could never have complete, up-to-date information on the state of the system. Therefore control must be distributed throughout the system so that local decisions can be made close to where they are needed*

(4) ***Inconsistent Information****. Information from outside the system or even from different parts of the same system may turn out to be inconsistent. Therefore decisions must be made by the components of an open system by considering whatever evidence is currently available*

(5) ***Arm's-length relationships****. The components of an open system are at en arm's- length relationship: the internal operation, organization, and state of one computational agent may be unknown and unavailable to another agent for reasons of privacy or outage of communications. Information should be passed by explicit communication between agents to conserve energy and maintain security. This ensures that each component can be kept simple since it only needs to keep track of its own state and its interfaces to other agents*

(6) ***Continuous operation****. Open systems must be reliable. They must be designed so that failures of individual components can be accommodated by operating components while the failed components are repaired or replaced*"[6].

A key concept of this definition is the information notion considered as input and output streams. This informational opening is an essential characteristic of Hewitt's open system definition. This point of view is in accordance with computers since their origin, i.e. systems of data processing. The internal architecture of these systems is not at all affected by this activity of production and consumption of data.

In completely different fields - physics and biology -, the scientist Ludwig von Bertalanffy developed in the 40s an open system theory [2]. Integrated into a General System Theory [3] an open system is characterized by a continual exchange of materials and energy, at the same time between the interior and the outside of the system, but

also between its internal components. This dynamics allows a living organism to grow, develop, reproduce and to survive in spite of external changing conditions, while causing regeneration, renewal, destruction and replacement of the entities which compose it. The exchange of constituents which cannot be reduced to information provides the entity with a type of opening of a completely different nature. Applied to the software agent field, this opening of physical nature offers more important flexibility than the informational opening but leads to a structural modification of the agent. This transformation must imperatively be controlled by equipping the agent with adapted internal mechanisms, under penalty of causing its stop.

The following section introduces a model of software agent which suits the study of the concept of physical opening.

2.2 Systemic agent

Thanks to its broad applicability, the systemic approach [12], [4] provides - in the software agent context - the abstract tools allowing the consideration of the two types of openings considered previously: the informational opening and the physical opening. A *systemic agent* is a software agent (1) surrounded by an environment: that means that the agent is located in a universe where there are action and reciprocal reaction, (2) provided with a software architecture which changes in time, (3) in order to carry out a certain activity.

This last point - the activity of the agent - must be clarified. Indeed, when it is considered within its most general framework, the activity of the agent has a double aspect: the functioning related to the task which was assigned to it and the functioning bound to the preservation of its physical integrity. Then, the agent has a double finality: an *external finality* materialized in the form of a task to be carried out or of a service to be supplied and an *internal finality* which object is to maintain the internal integrity of the agent. The designer of systemic agent is then confronted with the following problem: how to manage conflicts that may occur between the two finalities. In other words, the decoupling of the two finalities poses a problem of arbitration which must be resolved at the level of the internal architecture of the agent. The selected solution is to provide the agent with a two-layered architecture, a level associated with the external finality and a level associated with the internal finality. In addition, when the expression of the two finalities leads to a conflict, it is the internal finality which is privileged.

The principle of a dual architecture posed, we must now specify its content that is to define the nature of the elements which make it up: the properties of the agent.

3 Attributes and Qualities

Many definitions and characterizations of the concept of agent were published in the specialized literature [8], [13]. Certain studies articulate their analyses around two principal axes: properties possessed by the agent and the application area considered

[9]. Our previous works led us to keep only one of the two axes: properties. Two classes of properties are then distinguished: *attributes* and *qualities* [14].

3.1 Attributes

An attribute materializes a property of an agent which can be reduced to a mechanism, a perfectly known or customisable software device. For instance, the *mobility* attribute can be provided with a single parameter: the next site to be reached. Two pieces of information can customize the *replication* attribute: the rate of replication and the site of replication. The determination of the number and the meaning of the parameters of an attribute is made during the conception of the attribute independently from the agent which will contain it. The decoupling between the realization of the attribute and the various considered parameter settings led us to associate a mechanism and one or several use policies to every attribute.

Two main attributes will be used within the open agent's framework: *mobility* and *transformation*. The mobility attribute is present in many mobile agent based applications [11], and will be used in section 5.2 to illustrate an aspect of the physical opening. The second attribute is specific to the model of agents which we develop: the Systemion Model. This model will be presented in the following section. Two additional attributes will be evoked but not explained in detail: *replication* and *perception*.

Mobility: mobility allows an agent to migrate to other sites. The notion of itinerary is often associated to mobility. An itinerary corresponds to an ordered list of hosts having to be visited by the agent, possibly specifying the actions to be carried out in the case of an unreachable or hostile host. An itinerary is characterized by a site of departure, a succession of intermediate sites and a site of arrival; in many mobile agent applications the site of arrival is the same as the site of departure; it is then called circular itinerary. Sometimes, the notion of route is not adapted because it is preferable in certain cases, to calculate at the last moment the next site to be visited (to reach for example the least loaded host).

Modification: the modification attribute can be interpreted in various manners. We will distinguish two variations from this attribute: evolution and metamorphosis. *Evolution* can be seen as a succession of gradual transformations, relatively slow and going in the same direction. This type of transformation is very studied in several fields of research such as adaptive systems and artificial life. In our work, we are interested more particularly in the abrupt transformation of a software agent. This type of transformation, that we call *metamorphosis*, is defined as a change of nature or structure of the agent, more or less radical. Contrary to evolution, metamorphosis is a fast and sudden change.

3.2 Qualities

Unlike the notion of attribute, a quality cannot be given a precise definition accepted by all the researchers. Autonomy and intelligence are two paradigmatic examples of what we understand by qualities. Qualities are difficult to measure as they are manifold. Consequently, there are various complementary models for a quality: for instance, there are several models of autonomy. Indeed, there is a great variety of points of view among the researchers who are interested in this property: autonomy can be considered either as a global property or as a partial property of the agent, as a social or not social characteristic of the agent [5], [7]. Qualities introduce complexity within the agent.

To illustrate one of the aspects of the physical opening of an agent, we will use a model of autonomy which considers it as a partial and not social property of the agent in section 5. This model derives from the definition: *autonomy is the condition of a person or a group that chooses its own laws*. In our context, policies or behaviours take the place of laws.

The next section synthesizes the various concepts presented previously, within a particular architecture of agent.

4 The Systemion Model

The *systemion model* (contraction of *systemic daemon*) is a particular model of systemic agent [cf section 2.2]. A *systemion* [14] is a software agent which integrates in its architecture, the concepts of internal finality and external finality, qualities, attributes and task. This architecture can be split up into two subsystems:
(1) a *functional subsystem* which materializes the external finality, i.e. properties (qualities and attributes) 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. Application complexity (including cooperation, coordination and communication interactions) is embedded into the task abstraction. To be able to integrate the first task in its functional subsystem or to change current function, a systemion must incorporate into its behavioural subsystem, the metamorphosis attribute
(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. This software layer implements the internal finality (Figure 1).

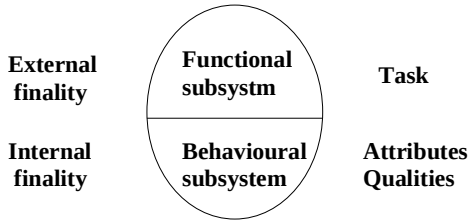

Figure 1. Systemion Architecture

4.1 Tasks

The functional subsystem establishes the current competence of the systemion in the form of a task. It is convenient to consider a task as being constituted of one or several components, each one providing a distinct service or all the components fulfilling a single function. Using the modification attribute included in the behavioural subsystem, the functional subsystem of a systemion can be modified by executing predefined operations on one or several components which it contains. Four operations on a component are defined: (1) *adding*: the component is inserted into the functional part of the systemion and activated; we shall say that the component is active (2) *removing*: it is the reverse operation of the previous one; the component is removed by the functional subsystem (3) *freezing*: the component remains present in the functional part of systemion but is not executed until a new reactivation takes place; the component is frozen (4) *Activation*: the component is reactivated (Figure 2).

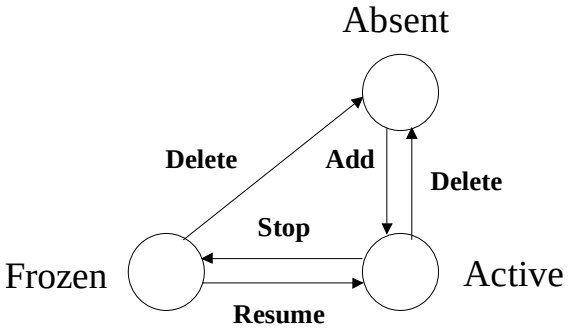

Figure 2. Task modification

Being able to dynamically add a component to a systemion contributes to extend the rendered services (extensibility). Dynamically replacing a component by another allows to adapt it to the particular characteristics of a given host or to modify the service to be provided. Finally, temporary freezing of a component allows a mobile agent not to execute a component on one or several visited sites, without deeply modifying the systemion structure. Indeed, addition and removing of components can be expensive operations in CPU time and bandwidth.

4.2 Using the Systemion Model

Using the Systemion Model, we will analyze two classes of very different agents: *system daemons* and *worms*. This analysis will enable us to illustrate the concept of physical opening in the following section.

Daemons are present in all modern computer systems. They carry out background service functions such as the management of network connections and printing requests. For that, they continuously await the requests coming from the client processes, then carry out the requested service. From a systemion point of view, daemons possess only a functional subsystem because the properties they implement are only dedicated to the execution of the task which was assigned to them: daemons are *unitask agents*. Deprived of internal finality, the behavioural subsystem of a daemon is empty (Figure 3.a).

Without simplifying excessively, we shall say that the purpose of a worm is to replicate itself locally or remotely using a communication network. According to the systemion architecture a worm is constituted by a behavioural subsystem which integrates two attributes: a mechanism of perception (local perception and perception of remote hosts) and a mechanism of replication (local replication and remote replication), attributes which confer a certain mobility to it. Not executing any particular task, a worm doesn't integrate a functional subsystem, which is empty and fixed. Indeed, its functional subsystem will not undergo any operational modification during its lifespan (Figure 3.b).

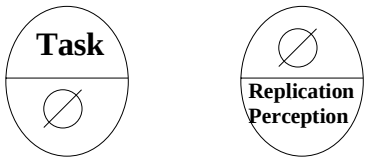

Figure 3. (a) Daemon (b) Worm

5 Physical opening

5.1 Open functional subsystem

The agents first studied were all functionally determined (empty or unitask). The systemion model supplies the tools necessary to the design and the implementation of agents functionally not determined. An agent of this type is provided with a functional subsystem which can vary in time, following the various tasks which it has to carry out. So that this dynamic modification of function can take place, the agent incorporates into its behavioural subsystem a suitable mechanism: the *modification* attribute presented in section 3.1. Not to jeopardize the systemion's existence the change of current task is triggered by the behavioural subsystem but the modifica-

tions are carried out in the functional subsystem. Thus it constitutes the most flexible part of the systemion.

Incorporating this internal transformation mechanism, the systemion can evolved in different ways. Let us illustrate this with an example. Figure 4 describes the following scenario: at time t a mobile agent arrives on host A. This systemion is an empty functionally not determined agent. It means that its functional subsystem does not contain any task to execute when host A receives it. At time t+1 the agent includes in its functional subsystem task T1 lying in a task repository. During its execution task T1 "asks" the systemion to migrate to site B. After the migration the systemion continues the execution of T1 on B, then deletes T1 from its functional subsystem and adds task T2 to it. These different operations are carried out using the ones presented in section 4.1. The systemion model enables to build *pluritask agents*.

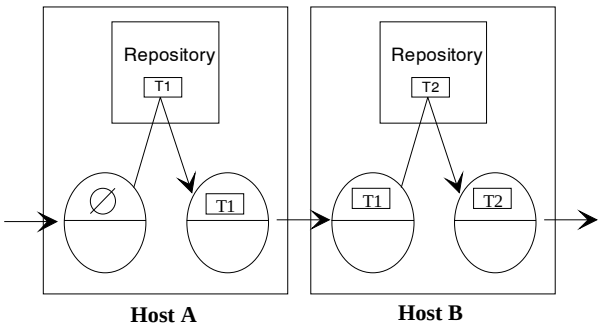

Figure 4. Agent functionally not determined

A pluritask agent can be used in various situations: (1) an empty mobile systemion is used to discover a network or to test its environment, (2) a task code used at a given moment, may be no longer accurate in a while since the main characteristics of the local environment may change, so the agent deletes this unusable task code.

5.2 Open behavioural subsystem

The physical opening can be applied to the behavioural subsystem of the agent. As we have already explained about the functional subsystem, the aim of the physical opening is adding or removing behaviours having an external origin. The fundamental difference from task changing is that the modification(s) concern the part of the agent that ensures its reliability, i.e. the subsystem maintaining the stability of the agent. The physical opening of the behavioural subsystem introduces a risk for the agent's life.

We shall now illustrate how this opening can be done using the concepts presented above. The behavioural subsystem includes the attributes and qualities which define the essence and personality of the agent which are independent from the tasks that may be given to it. The concept of attribute differentiates the mechanism from the modalities of its use; each modality is called use policy or more simply policy. A

particular model of autonomy is the autonomy with regard to an attribute. This model can be described as follows: : an agent is *autonomous with regard to an attribute* if it can rationally choose or randomly elect one policy among several ones associated to this attribute and can change current policy during its life.

In other words, an agent is autonomous with regard to an attribute if the policy it executes is not always explicitly forced on it by the environment. This kind of autonomy requires a set of behaviours and a decision module used by the agent to select the current policy (Figure 5). We do not intend to present the structure of the choice module in details. That is why it is represented by a black box. When a new external behaviour must be incorporated into the agent, it automatically results in modification within the choice module. It means that the choice module also has to be modified, even totally replaced (by using the physical opening).

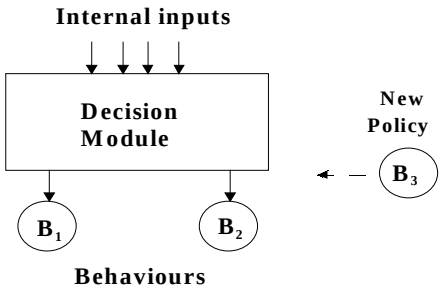

Figure 5. An incoming new behaviour

If we take mobility as an example of attribute, different migration policies are possible: (1) random navigation, (2) navigation directed according to a special criterion (e.g. a machine load). A mobile agent will include in its behavioural subsystem a choice module adapted to the handling of these two policies. If a new behaviour has to be taken into account by the agent, e.g. circular navigation, the choice module will also have to be modified. It can have various more or less serious consequences since policies have different complexities. Indeed, a behaviour can be made of one action: directed navigation and random navigation are mono-action policies. On the contrary, circular navigation is made of several elementary moves.

The physical opening of the behavioural subsystem is trickier than the physical opening of the functional subsystem. Anyway, the systemion model makes it possible to study both of them.

6 Conclusion

Complex systems and open systems have common characteristics that theories resulting from the systemic thought try to specify. By applying to the software agent the concepts of physical and informational opening and of two-layered architecture with

stable/unstable contents, the Systemion Model has illustrated the interest of the dual reasoning in the study of complex computer systems.

References

1. Auyang, S. Y.: Foundations of Complex-System Theories in Economics, Evolutionary Biology, And Statistical Physics. Cambridge (UK), (1998).
2. Bertalanffy, L. von: The Theory of Open Systems in Physics and Biology. Science, 111 (1950), pp. 23-29.
3. Bertalanffy, L. von: General System Theory. George Braziller, New York, (1968).
4. Eriksson, D. M.: A Principal Exposition of Jean-Louis Le Moigne's Systemic Theory. Cybernetics and Human Knowing, Vol. 4, no 2-3, (1997).
5. Franklin, S., Graesser, A.: 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. Hewitt, C.: Offices Are Open Systems. ACM Transactions on Office Information Systems, Vol. 4, No.3, July 1986, pp. 271-287, (1986).
7. Hexmoor, H., Castelfranchi, C., Falcone, R. (eds.): Agent Autonomy. Kluwer Academic Publishers, Boston, (2003).
8. Jennings, N., Wooldridge, M.: Applications of Intelligent Agents. In *Agent Technology: Foundations, Applications, and Markets* (Edited by N. R. Jennings and M. Wooldridge) Springer Computer Science, (1998).
9. Jennings, N., Sycara, K., Wooldridge, M.: A Roadmap of Agent Research and Development. In *Autonomous Agents and Multi-Agent Systems*, 1, pp 275-306, Kluwer Academic Publishers, Boston, (1998).
10. Kephart, J. O., Chess, D. M.: The Vision of Autonomic Computing. IEEE Computer, Vol. 36, no 1, pp 41-50, (2003).
11. Milojicic, D.: Trend Wars – Mobile agent applications. *IEEE Concurrency*, Sept 1999, pp 80-90, (1999).
12. Le Moigne, J. L.: La théorie du système général – Théorie de la modélisation –. Ed. PUF, Paris, (1977).
13. Parunak, H. V. D., Breuckner, S., Sauter, J.: A Preliminary Taxonomy of Multi-Agent Interaction. In *Agent-Oriented Software Engineering (AOSE) IV*, P. Giorgini, Jörg Müller, James Odell, eds., Lecture Notes on Computer Science volume (forthcoming), Springer, Berlin, (2004).
14. Sanchis, E.: 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).
15. Simon, H. A.: The Sciences of the Artificial. MIT Press, Cambridge (Massachusetts), (1996).
16. Wolfram, S.: Universality and Complexity in Cellular Automata. Physica D, 10, Jan 1984, pp 1-35, (1984).