# Basic approach to Emergent Programming - Feasibility Study for Engineering Adaptive Systems using Self-Organizing Instruction-agents

Jean-Pierre Georgé, Marie-Pierre Gleizes, Pierre Glize

# Basic approach to Emergent Programming

# Feasibility Study for Engineering Adaptive Systems using Self-Organizing Instruction-agents

Jean-Pierre Georgé, Marie-Pierre Gleizes, Pierre Glize

IRIT, Université Paul Sabatier, 118 route de Narbonne, 31400 Toulouse, France
{george, gleizes, glize}@irit.fr

**Abstract.** We propose to investigate the concept of an Emergent Programming Environment enabling the development of complex adaptive systems. This is done as a means to tackle the problems of the growth in complexity of programming, increasing dynamisms in artificial systems and environments, and the lack of knowledge about difficult problems and their solutions. For this we use as a foundation the concept of *emergence* and a multi-agent system technology based on cooperative self-organizing mechanisms.
The general objective is then to develop a complete programming language in which each instruction is an autonomous agent trying to be in a cooperative state with the other agents of the system, as well as with the environment of the system. By endowing these *instruction-agents* with self-organizing mechanisms, we obtain a system able to continuously adapt to the task required by the programmer (i.e. to program and re-program itself depending on the needs). The work presented here aims at showing the feasibility of such a concept by specifying, and experimenting with, a core of *instruction-agents* needed for a sub-set of mathematical calculus.

## 1 Introduction

In the last few years, the use of computers has spectacularly grown and classical software development methods run into numerous difficulties. Operating systems are a good example of extremely complex software which are never exempt of problems. The classical approach, by decomposition into modules and total control, cannot guaranty the functionality of the software given the complexity of interaction between the increasing and variable number of modules, and the shear size of possibilities. Adding to this, the now massive and inevitable use of network resources and distribution only increases the difficulties of design, stability and maintenance.

### 1.1 Neo-computation Problems

This state is of interest to an increasing number of industrials, including IBM who wrote in a much relayed manifesto : "*Even if we could somehow come up with*

*enough skilled people, the complexity is growing beyond human ability to manage it. Pinpointing root causes of failures becomes more difficult, while finding ways of increasing system efficiency generates problems with more variables than any human can hope to solve. Without new approaches, things will only get worse*" [14].

These kind of applications are what we call *neo-computation problems*, namely: autonomic computing, pervasive computing, ubiquitous computing [18], emergent computation, ambient intelligence, amorphous computing... This set of problems have in common the inability to define the global function to achieve, and by consequence to specify at the design phase, a derived evaluation function for the learning process. They are characterized by :

- a great number of interacting components (intelligent objects, agents, software);
- a variable number of these components during runtime (open system);
- the impossibility to impose a global control;
- an evolving and unpredictable environment;
- a global task to achieve.

### 1.2 Problem Solving by Emergence

Given the previous characteristics, the challenge is to find new approaches to conceive these new systems by taking into account the increasing complexity and the fact that we want reliable and robust systems. For this, because of the similarities, it seems opportune to look at natural systems - biological, physical or sociological - from an artificial system builder's point of view so as to understand the mechanisms and processes which enable their functioning.

In Biology for example, a lot of natural systems composed of autonomous individuals exhibit aptitudes to carry out complex tasks without any global control. Moreover, they can adapt to their surroundings either for survival needs or to improve the functioning of the collective. This is the case for example in social insects colonies [4] such as termites and ants [3]. The study of swarm behaviours by migratory birds or fish shoals also shows that the collective task is the result of the interactions between autonomous individuals. Non supervised phenomena resulting from the activity of a huge number of individuals can also be observed in human activities such as the synchronization of clapping in a crowd or traffic jams. But the most surprising is still the appearance of human consciousness out of the chemical and electrical jumble of our brain.

There is a common factor among all theses systems : the emergent dimension of the observed behaviour. Thus it is quite legitimate to study emergence so as to understand its functioning or at least to be able to adequately reproduce it for the design of artificial systems. This would enable the development of more complex, robust and adaptive systems, needed to tackle the difficulties inherent to *neo-computation* problems. In this way, interesting and useful emergent phenomena will be used in artificial systems when needed. Contrariwise, they will still appear sooner or later the more complex the systems are getting but will be unexpected

and unwanted. To prevent this, one orientation would be, in our opinion, that the scientific community studies and develops new theories based upon emergence. The prerequisites of such a theory could be resumed in four points :

- to start from the Systems Theory field;
- to focus on the parts of the system and their functioning;
- to depend neither from the systems finality, nor its environment (there can still be constraints or some form of feedback but there should be no imposed behaviour for the system);
- to be independent from the material support into which a given system will be incarnated (biological, technological, ...) : it has to be generic;

It is noteworthy that some research is already being done for quite some years now to bring emergence into artificial systems, but it is still very localized. For example, the *Santa Fe Institute* has acquired an international renown for its works on complexity, adaptive complex systems and thus emergence. These are also the preoccupations of *Exystence*, the European excellence network on complex systems, or the recently begun *ONCE-CS*, the Open network of Centres of Excellence in Complex Systems.

### 1.3 Going to the Lowest Level : the Instructions

If we suppose that we can manage to use the emergent phenomena to build artificial systems, this will be by specifying the behaviour of the parts of the systems so that it will enable their interactions to produce the expected global emergent behaviour of the system. A relevant question would be to ask about what parts we are focusing on and on which level. As with classical software engineering, any decomposition could be interesting, depending on the nature of the system being build.

We propose here to focus on the lowest possible level for any artificial system : the instruction level. We will explain our theoretical and experimental exploration of the concept of *Emergent Programming*. This concept is explained in the next section (section 2). Its use relies on emergence and self-organization (section 5) on one hand, and on a multi-agent approach called *AMAS* (Adaptive Multi-Agent System)[11] (section 3) on the other hand. A sub-problem which we called the *elementary example* has been thoroughly explored and is presented in section 4 where we then show how the learned lessons can lead us forward in our exploration of *Emergent Programming* and more generally of problem solving using emergence.

## 2 Emergent Programming

### 2.1 The Concept

In its most abstract view, *Emergent Programming* is the automatic assembling of instructions of a programming language using mechanisms which are not explicitly informed of the program to be created. We may consider that for a

programmer to produce a program comes down to finding which instructions to assemble and in which precise order. This is in fact the exploration of the search space representing the whole set of possible programs until the right program is found. However, if this exploration is easy when the programmer has a precise knowledge about the program he wants and how to obtain it, it grows more and more difficult with the increase of complexity of the program, or when the knowledge about the task to be executed by the program becomes imprecise or incomplete. Then are we not able to conceive an artificial system exploring efficiently the search space of the possible programs instead of having the programmer do it ? Only very few works exists on this topic. One noteworthy try has been done by Koza using Genetic Algorithms and a LISP language [16], but the main hindrance of GA is the need for a specific evaluation function for each problem, which can be very difficult to find. At the opposite, we aim at an as generic as possible approach.

To approach the problem of *Emergent Programming* concretely, we chose to rely on an adaptive multi-agent system using self-organizing mechanisms based on cooperation as it is described in the *AMAS* theory. This theory can be considered as a guide to endow the agents with the capacity to continuously self-organize so as to always tend toward cooperative interactions between them and with the environment. It then claims that a cooperative state for the whole system implies the functional adequacy of the system, i.e. that it exhibits a behaviour which satisfies the constraints of the parts of the system as well as from the environment (e.g. a user).

### 2.2 The Instruction-Agents

In this context, we define an agent as an instruction of a programming language. Depending on the type of the instruction he is representing, the agent possesses specific competences which he will use to interact with other *instruction-agents*. A complete program is then represented by a given organization of the *instruction-agents* in which each agent is linked with partners from which he receives data and partners to which he sends data. The counterpart of the execution of a classical program is here simply the activity of the multi-agent system during the exchange of data between the agents.

### 2.3 The Reorganization Process

We can now appreciate all the power of the concept : a given organization codes for a given program, and thus, changing the organization changes the final program. It comes down to having the agents self-organize depending on the requirements from the environment so as to continuously tend toward the adequate program (the adequate global function). In principle, we obtain a system able to explore the search space of the possible programs in place of the programmer. Everything depends on the efficiency of the exploration to reach an organization producing the right function. An important part of our work on *Emergent Programming* has been the exploration of the self-organization mechanisms which

enable the agents to progress toward the adequate function, depending on the constraints of the environment but without knowing the organization to reach or how to do it (since this is unknown for the problems we are interested in).

### 2.4 A Neo-programming Environment

The system will not be able to grow *ex nihilo* all by itself, all the more if we want to obtain higher level programs (programs with more complex behaviours). As the programmer with his classical programming environment, the *neo-programmer* will have to affect the development of the system through a *neo-programming environment*, at least at the beginning. It is a matter of supplying the tools to shape the environment of the system so as to have this environment constrain the system toward the adequate function. But in a pure systems theory's view, the *neo-programmer* is simply part of the environment of the system.

But the *neo-programming environment* will certainly have to be more than a simple envelope for the developing system. We will probably need to integrate some tools for the observation of the evolution of the system, means to influence this evolution, the type and proportions of *instruction-agents*, to affect some aspects of the structure. Moreover, a complex program is generally viewed as a modular construct and the *neo-programmer* may want to influence this modular structure, either by manipulating some sorts of "bricks", each being an *emergent programming* system, or by letting these "bricks" self-organize in the same manner as their own components.

At the end, we will obtain a system able not only to "find" *how* to realize the adequate function, but also to continuously adapt to the environment in which it is plunged, to react to the strongly dynamic and unpredictable nature of real world environments, and all this by presenting a high grade of robustness. Indeed, because of its nature, the system would be able to change its internal structure any time and by consequence its performed function, or even grow by adding instructions to respond to some partial destruction or to gain some new competences.

The research we did on *Emergent Programming* was to explore the feasibility of the concept. For this, we restrained the programming language to the instructions needed for a subset of mathematical calculus, of which the *elementary example* (section 4) is a representative. We specified such a core of agents and put it through experimentation. For this an environment has been implemented : *EPE* (*Emergent Programming Environment*) [9]. These experimentations enabled us to explore different self-organization mechanisms for the *instruction-agents* so as to find those who lead to the emergence of the adequate function. Part of these mechanisms are described here.

## 3 Using Cooperative Agents as the Engine for Self-organization

### 3.1 Adapt the System by its Parts

We consider that each part $P_i$ of a system S achieves a partial function $f_{pi}$ of the global function $f_s$ (Figure 1). $f_s$ is the result of the combination of the partial functions $f_{pi}$, noted by the operator "o". The combination being determined by the current organization of the parts, we can deduce $fs = f_{p1} \, o \, f_{p2} \, o \, ... \, o \, f_{pn}$. As generally $f_{p1} \, o \, f_{p2} \neq f_{p2} \, o \, f_{p1}$, by transforming the organization, the combination of the partial functions is changed and therefore the global function $f_s$ changes. This is a powerful way to adapt the system to the environment. A pertinent technique to build this kind of systems is to use adaptive multi-agent systems. As in Wooldridge's definition of *multi-agent systems* [19], we will be referring to systems constituted by several autonomous agents, plunged in a common environment and trying to solve a common task.

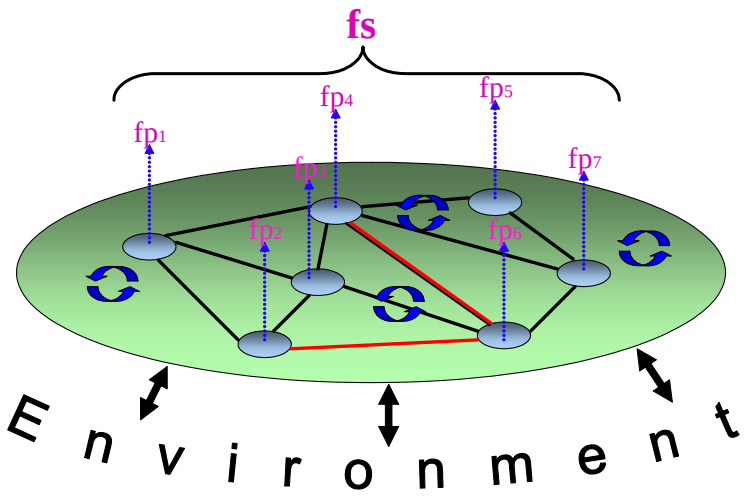


**Fig. 1.** Adaptation : changing the function of the system by changing the organization.

### 3.2 The Theorem of Functional Adequacy

Cooperation was extensively studied in computer science by Axelrod [1] and Huberman [15] for instance. "*Everybody will agree that co-operation is in general advantageous for the group of co-operators as a whole, even though it may curb some individual's freedom*" [12]. Relevant biological inspired approaches using cooperation are for instance *Ants Algorithms* [7] which give efficient results in

many domains. In order to show the theoretical improvement coming from cooperation, we have developed the *AMAS* (Adaptive Multi-Agent System)[11] theory which is based upon the following theorem. This theorem describes the relation between cooperation in a system and the resulting functional adequacy [1] of the system.

**Theorem**. *For any functionally adequate system, there exists at least one cooperative internal medium system that fulfils an equivalent function in the same environment.*

**Definition**. *A cooperative internal medium system is a system where no* Non-Cooperative Situations *exist.*

**Definition**. *An agent is in a Non-Cooperative Situation (*NCS*) when : (1) a perceived signal is not understood or is ambiguous; (2) perceived information does not produce any activity of the agent; (3) the conclusions are not useful to others.*

### 3.3 Consequence

This theorem means that we only have to use (and hence understand) a subset of particular systems (those with cooperative internal mediums) in order to obtain a functionally adequate system in a given environment. We concentrate on a particular class of such systems, those with the following properties [11]:

- The system is cooperative and functionally adequate with respect to its environment. Its parts do not 'know' the global function the system has to achieve via adaptation.
- The system does not have an explicitly defined goal, rather it acts using its perceptions of the environment as a feedback in order to adapt the global function to be adequate. The mechanism of adaptation is for each agent to try and maintain cooperation using their skills, representations of themselves, other agents and environment.
- Each part only evaluates whether the changes taking place are cooperative from its point of view - it does not know if these changes are dependent on its own past actions.

This way of engineering systems has been successfully applied on numerous applications with very different characteristics for the last ten years (autonomous mechanisms synthesis[6], flood forecast[10], electronic commerce and profiling,...). On each, the local cooperation criterion proved to be relevant to tackle the problems without having to resort to an explicit knowledge of the goal an how to reach it.

[1] "Functional" refers to the "function" the system is producing, in a broad meaning, i.e. what the system is doing, what an observer would qualify as the behaviour of a system. And "adequate" simply means that the system is doing the "right" thing, judged by an observer or the environment. So "functional adequacy" can be seen as "having the appropriate behaviour for the task".

### 3.4 The Engine for Self-organization

The designer provides the agents with local criterion to discern between cooperative and non-cooperative situations. The detection and then elimination of NCS between agents constitute the engine of self-organization. Depending on the real-time interactions the multi-agent system has with its environment, the organization between its agents emerges and constitutes an answer to the aforementioned difficulties of *neo-computation problems* (indeed, there is no global control of the system). In itself, the emergent organization is an observable organization that has not been given first by the designer of the system. Each agent computes a partial function $f_{pi}$, but the combination of all the partial functions produces the global emergent function fs. Depending on the interactions between themselves and with the environment, the agents change their interactions i.e. their links. This is what we call self-organization.

By principle, the emerging purpose of a system is not recognizable by the system itself, its only criterion must be of strictly local nature (relative to the activity of the parts which make it up). By respecting this, the *AMAS* theory aims at being a theory of emergence.

## 4 The Elementary Example

We tried to find an *emergent programming* system as simple as possible (i.e. with the smallest number of agents with the simplest functioning), but still needing reorganizations so as to produce the desired function. The advantages of such a case study are that it is more practical for observation, that it leads to less development complexity and that it presents a smaller search space.

### 4.1 Description

The specification of each agent depends on the task he has to accomplish, of his "*inputs*" and "*outputs*". The agents communicate by messages but to accomplish the actual calculation, we can consider that the agents are expecting values as inputs to be able to provide computed values as outputs. Schematically, we can consider exchanges between agents as an electronic cabling between outputs and inputs of agents.

The elementary example we choose is constituted of 6 agents : 3 "*constant*" agents, an "*addition*" agent, a "*multiplication*" agent and an "*output*" agent. A "*constant*" agent is able to provide the value which has been fixed at his creation. The 3 the system contains have been given sufficiently different values so as to prevent calculation ambiguity : *AgentConstantA* (value = 2), *AgentConstantB*(value = 10) and *AgentConstantC* (value = 100). Combined with *AgentAddition* and *AgentMultiplication*, the values produced by the system are results from organizations like $(A+B)*C$ or any other possible combination. *AgentOut* simply transmits the value he receives to the environment. But he is also in charge of retrieving the feedback from the environment and forward it into the system.

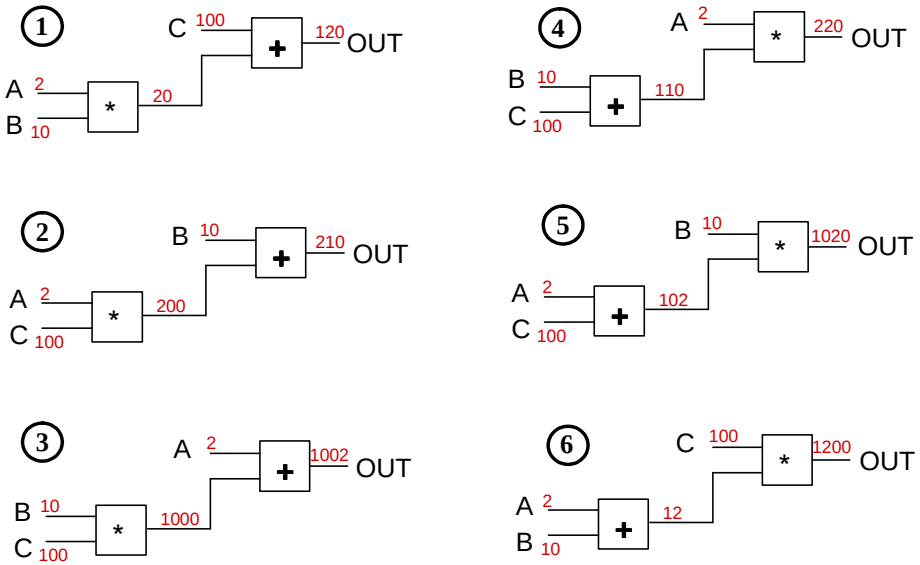


**Fig. 2.** The 6 different possible types of functional organizations for the elementary example.

The size of the complete search space is $6^5$, that is 7776 theoretically possible organizations, counting all the incomplete ones (i.e. where not every agent has all his partners). There are 120 complete organizations and among those, 24 are functional (they can actually calculate a value) if we count all the possible permutations on the inputs which do not change the calculated value. In the end, we have 6 types of different organization (cf. Figure 2) producing these 6 values : 120, 210, 220, 1002, 1020 and 1200. The aim is to start without any partnerships between agents and to request that the system produces the highest value for example.

### 4.2 Reorganization Mechanisms

In accordance with the *AMAS* theory, the agent's self-organizing capacity is induced by their capacity to detect NCS (Non-Cooperative Situations), react so as to resorb them and continuously act as cooperatively as possible. This last point implies in fact that the agent also has to try to resorb NCS of other agents if he is aware of them: to ignore a call for help from another agent is definitely not cooperative. We will illustrate this with the description of two NCS and how they are resorbed.

**Detection**

*NCSNeedIn* : the agent is missing a partner on one of his inputs. Since to be cooperative in the system he has to be useful, and to be useful he has to be able to compute his function, he has to find partners able to send values toward his input.

Most NCS lead the agent to communicate so as to find a suitable (new) partner. These calls, because the agents have to take them into account, also take the shape of NCS.

*NCSNeedInMessage* : the agent receives a message informing him that another agent is in a *NCSNeedIn* situation.

**Resorption**

*NCSNeedIn* : this is one of the easiest NCS to resorb because the agent only has to find any agent for his missing input. And the agents are potentially always able to provide as many values on their outputs for as many partners as needed. The agent has simply to be able to contact some agent providing values of the right type (there could be agents handling values of different types in a system), i.e. corresponding to his own type. So he generates a *NCSNeedInMessage* describing his situation (his needs) and send it to his acquaintances (because they are the only agents he knows).

*NCSNeedInMessage* : the agent is informed of the needs of the sender of the NCS and his cooperative attitude dictates him to act. First, he has to judge if he is relevant for the needs of the sender, and if it is the case, he has to propose himself as a potential partner. Second, even if he is not himself relevant, one of its acquaintances may be. He will do what the *AMAS* theory calls a resorption by restricted propagation : he tries to counter this NCS by propagating the initial message to some acquaintances he thinks may be the most relevant.

For each NCS the agent is able to detect (there are 10 NCS in total for these agents), a specific resorption mechanism has been defined. It is a precise description of the decision making of the agent depending on his state and on what it perceives. For other NCS, the mechanisms become quite complicated, and require a long description. For an exhaustive presentation, please refer to [9].

These NCS and their symmetric for a missing partner on an output enable the system to produce an organization where each agent has all his needed partners. To obtain the functional adequacy for the system means that the final organization is able to produce the expected result. The main question is how to introduce mechanisms in the resorption of the NCS to enable the agents as a whole to reach this organization. For this, they need some kind of "*direction*" (but on local criterion) to get progressively closer to the solution, a local information to judge this proximity. The information used here is simply a "smaller/bigger" feedback type that the environment sends to the system and that will be dispatched between the agents by propagation and by taking other the goal (smaller or bigger). The agent then tries to satisfy its new goal and staying at the same time the most cooperative possible with the other agents. This will bring the system as a whole to produce a smaller or bigger value.

Of course, the agents will get into conflict with other agents when trying to reach these goals and the self-organizing mechanisms take that into account. Each agent also manipulates a knowledge about the prejudice he inflicts or may inflict following changes he induces in the organization. For this, the agents communicate to each other, when necessary, their current goal and state. When

choosing a new partner an agent takes into account the impact of its decision upon the concerned agents, i.e which agents will be hindered from reaching their goal, which agents will be in a worse state than before and in what proportion. By minimizing these prejudices (which is a form of cooperation), the whole organization progresses.

It is important to note that the information which is given as a feedback is not in any way an explicit description about the goal and *how* to reach it. Indeed, this information does not exist : given a handful of values and mathematical operators, there is no explicit method to reach a specific value even for a human. They can only try and guess, and this is also what the agents do. That is why we believe the resolution we implemented to be in the frame of emergence.

### 4.3 Results and Discussion

**Results** The elementary example has been implemented in Java as a multi-threaded agent platform able to run any type of instruction-agents. It also simulates the environment for an organization of instruction-agents and provides tools for the observation and analysis of the reorganization process.

First of all, the internal constraints of the system are solved very quickly : in only a few reorganization moves (among the 7776 possible organizations), all the agents find their partners and a functional organization is reached. Then, because the system is asked to produce the highest value for example (configuration 6, Figure 2), other NCS are produced and the system starts reorganizing toward its goal. In accordance with the AMAS theory, the system is considered to provide an adequate behaviour when no more NCS are detected (the environment is satisfied by the produced results).

On a few hundred simulations, the functional adequacy is reached in a very satisfactory number of organization changes. Since the search space if of 7776 possible organizations, a blind exploration would need an average of 3.888 checked organizations to reach a specific one. Since a functional organization possesses 4 identical instances for a given value (by input permutations), we would need 972 tries to get the right value. Experimentation shows that, whatever the initial organization (without any links or one of the 6 functionals), the system needs to explore less than a hundred organizations among the 7776 to reach one of the 4 producing the highest value. We consider that this self-organization strategy allows a relevant exploration of the search space. A noteworthy result is also that whatever organization receives the feedback for a better value, the next organization will indeed produce a better value (if it exists).

**Emergent Programming : A Universal Tool** If we define all the agents needed to represent a complete programming language (with agents representing variables, allocation, control structures, ...) and if this language is extensive enough, we obtain maximal expressiveness : every program we can produce with current programming languages can be coded as an organization of *instruction-agents*. In its absolute concept, *Emergent programming* could then solve any

problem, given that the problem can be solved by a computer system. Of course, this seems quite unrealistic, at least for the moment.

**Problem Solving using Emergence** But if we possess some higher-level knowledges about a problem, or if the problem can be structured at a higher level than the instruction level, then it is more efficient and easier to conceive the system at a higher level. This is the case for example when we can identify entities of bigger granularity which therefore have richer competences and behaviours, maybe adapted specifically for the problem.

Consequently, we will certainly be able to apply the self-organizing mechanisms developed for Emergent Programming to other ways to tackle a problem. Indeed, *instruction-agents* are very particular by the fact that they represent the most generic type of entities and that there is a huge gap between their functions and the function of a whole program. The exploration of the search space, for entities possessing more information or more competences for a given problem can only be easier. In the worst case, we can always try to use Emergent Programming as a way to specify the behaviour of higher-level entities (recursive use of emergence).

Let us consider for instance the problem of ambient intelligence : in a room, a huge number of electronic equipments controlled each by an autonomous microchip have as a goal the satisfaction of the users moving around it from day to day. The goal itself, user satisfaction, is really imprecise and incomplete, and the way to reach it even more. We claim that this problem is an ideal candidate for a problem solving by emergence approach: let us endow the entities with means to find by themselves the global behaviour of the system so as to satisfy the users. The challenge is to define the "right" self-organizing behaviours for the different equipments for them to be able to modify the way they interact to take into account the constraints of every one of them plus the external stimuli from the users (order, judgement, behaviour, ...). And we are convinced that this can only be done if the self-organization mechanisms tightly fit the frame of emergence.

## 5 Emergence and Self-organization

If we study specialized literature on emergence or self-organization, we can see that these are tightly linked. Yet, at the same time, we can see a lot of works focusing exclusively on the second without any mention, or only a brief, about the first. One explanation could be that the notion of emergence is quite abstract, even philosophical, making it difficult to fully grasp and therefore delicate to manipulate. At the opposite, self-organization is more concrete by its description in terms of mechanisms and thus, more easily used. But by concentrating solely on the mechanisms, are we not taking the risk to leave the frame of emergence? We give here a description of self-organization integrating emergence.

### 5.1 What is Self-organization ?

The self-organization field has from the very beginning tried to explore the internal mechanisms of systems producing emergent phenomena[2]. They tried to find the general functioning rules explaining the growth and evolution of the observed systemic structures, to find the shapes the systems could take, and finally to produce methods to predict the future organizations appearing out of changes happening at the component level of the systems. And these prospective results had to be applicable on any other system exhibiting the same characteristics (search for generic mechanisms). There are abundant definitions and descriptions of characteristics of emergence and self-organization in literature. We can sum it up as this:

**Definition**. *Self-organization is the set of processes within a system, stemming from mechanisms based on local rules which lead the system to produce structures or specific behaviours which are not dictated by the outside of the system* [8][13][17].

### 5.2 Understanding Self-organization

In most definitions about emergence and self-organization, there is the notion under some form or another of strictly local rules and resulting behaviours. There is also the strong constraint for these behaviors not to be imposed, dictated, explicitly informed or constrained by the environment of the system. The local character of a rule gives a strict and clear framework. But concerning the influence of the environment on the system, being it directly or through some internal rules, the exact characterization of this influence can be particularly difficult and vague.

Let us take the example of Bénard convection cells which is a classical example of self-organization. The phenomenon produced by self-organization is here the shape of the movement of water molecules which creates these particular observable flux structures (when looking top down at water just before it starts boiling, we can see hexagonal cells covering the bottom). The local rules are here the movement and collisions of the molecules. The fact that the molecules move more easily when they move in the same direction (because of less collisions) creates circulation fluxes. But the surfaces of the container as well as the influx of heat which forces the molecules to move are indeed part of the environment of the system and influence the behaviour of the system. We then have to decide of the impact and nature of this influence on the behaviour of the molecules and system. We can argue that it is indeed this influx of heat which compels the molecules to move and that the surfaces of the container also strongly constrain these movements. But this is not enough to explain *how* the molecules have to move, only that they have to, and in a given border. It is indeed the local collision rules which lead to the emergence of the hexagonal cells. The frame of self-organization seems here relatively clear after analysis but could have been argued against at the beginning.

In fact, in many cases the environment dictates very strong templates for the system to follow. Even if these templates are followed at a local level by autonomous entities, the more strong and precise they are, the less we think we can pretend to be in a self-organization frame. When wanting to use self-organization as the internal mechanism of an artificial system, we have to keep a critical attention on the influence the environment has on the system.

### 5.3 Using Emergence in Artificial Systems

Our work in this domain during the last decade lead us to give a "technical" definition of emergence in the context of multi-agent systems, and therefore with a strong computer science colouration. It is based on three points: what we want to be emergent, at what condition it is emergent and how we can use it [5].

1. **Subject**. The goal of a computational system is to realize an adequate function, judged by a relevant user. It is this function (which may evolve during time) that has to emerge.
2. **Condition**. This function is emergent if the coding of the system does not depend on the knowledge of this function. This coding has to contain the mechanisms facilitating the adaptation of the system during its coupling with the environment, so as to tend toward an adequate function.
3. **Method**. To change the function the system only has to change the organization of its components. The mechanisms which allow the changes are specified by self-organization rules providing autonomous guidance to the components' behaviour without any explicit knowledge of the collective function nor how to reach it.

## 6 Conclusion

We aimed at studying the feasibility of the concept of *Emergent Programming* by using self-organizing *instruction-agents*. We presented in this paper the concept and how we studied it. For this, we first described the frame of self-organization and emergence as we think can be applied in artificial systems. Then we described a generic approach for adaptive systems based upon a multi-agent system where the agents are endowed with self-organizing mechanisms based upon cooperation and emergence.

An elementary example has been used as a case study. Its implementation, and experimentation with, lead to the definition of the self-organizing mechanisms of the *instruction-agents* so as to enable them to make the system reach a given goal.

This study has been an interesting work to explore self-organization in MAS when confronted to difficult problems that we are persuaded need an Emergent solution. We claim that this approach would be really relevant for *neo-computation* problems such as ambient intelligence, if not directly with *instruction-agents*, by using the same kind of cooperative self-organization mechanisms.

## References


1. R. Axelrod. *The Evolution of Cooperation.* Basic Books, New York, 1984.
2. P. Ball. *The Self-Made Tapestry.* Oxford Press, 1999.
3. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence - from natural to artificial systems.* Oxford University Press, 1999.
4. S. Camazine, J.-L. Deneubourg, N. Franks, J. Sneyd, and E. Theraulaz, G.and Bonabeau. *Self-organization in biological systems.* Princeton University Press, 2002.
5. D. Capera, J. Georgé, M.-P. Gleizes, and P. Glize. Emergence of organisations, emergence of functions. In *AISB'03 symposium on Adaptive Agents and Multi-Agent Systems*, April 2003.
6. D. Capera, M.-P. Gleizes, and P. Glize. Mechanism type synthesis based on self-assembling agents. *Journal on Applied Artificial Intelligence*, 18(8-9), 2004.
7. M. Dorigo and G. Di Caro. *The Ant Colony Optimization Meta-Heuristic.* McGraw-Hill, 1999.
8. J. Georgé, B. Edmonds, and P. Glize. *Self-organizing adaptive multi-agent systems work*, chapter 16, pages 321–340. Kluwer Publishing, 2004.
9. J.-P. Georgé. *Résolution de problèmes par émergence - Étude d'un Environnement de Programmation Émergente.* PhD thesis, Université Paul Sabatier, Toulouse, France, 2004. http://www.irit.fr/SMAC/EPE.html.
10. J.-P. Georgé, M.-P. Gleizes, P. Glize, and C. Régis. Real-time simulation for flood forecast: an adaptive multi-agent system staff. In D. Kazakov, D. Kudenko, and E. Alonso, editors, *Proceedings of the AISB'03 symposium on Adaptive Agents and Multi-Agent Systems(AAMAS'03)*, pages 109–114, University of Wales, Aberystwyth, April 7-11 2003. SSAISB.
11. P.-P. Gleizes, V. Camps, and P. Glize. A theory of emergent computation based on cooperative self-oganization for adaptive artificial systems. In *Fourth European Congress of Systems Science*, Valencia, Spain, 1999.
12. F. Heylighen. Evolution, selfishness and cooperation; selfish memes and the evolution of cooperation. *Journal of Ideas*, 2(4):70–84, 1992.
13. F. Heylighen. *Encyclopedia of Life Support Systems*, chapter The Science of Self-organization and Adaptivity. EOLSS Publishers Co. Ltd, 2001.
14. P. Horn. Autonomic computing - ibm's perspective on the state of information technology. http://www.ibm.com/research/autonomic, 2001.
15. B. Huberman. *The performance of cooperative processes.* MIT Press / North-Holland, 1991.
16. J. R. Koza. Evolution and co-evolution of computer programs to control independently-acting agents. In *From animals to animats : proceedings of the first international conference on Simulation of Adaptative Behavior (SAB).* MIT Press, 1991.
17. I. Prigogine and G. Nicolis. *Self Organization in Non-Equilibrium Systems.* J. Wiley and Sons, New York, 1977.
18. M. Weiser and J. S. Brown. Designing calm technology. *PowerGrid Journal*, 1(1), 1996.
19. M. Wooldridge. *An introduction to multi-agent systems.* John Wiley & Sons, 2002.