Experiments in Emergent Programming Using Self-organizing Multi-Agent Systems

Jean-Pierre Georgé, Marie-Pierre Gleizes

Experiments in Emergent Programming Using Self-organizing Multi-agent Systems

Jean-Pierre Georgé and Marie-Pierre Gleizes

IRIT, Université Paul Sabatier, 118 route de Narbonne, 31062 Toulouse cedex, France
{george, gleizes}@irit.fr

Abstract. We propose to investigate the concept of an Emergent Programming Environment enabling the development of complex adaptive systems. 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. 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. 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.

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. [...]increasing system efficiency generates problems with more variables than any human can hope to solve. Without new approaches, things will only get worse*" [9]. Their answer to that is a scientific challenge they call *autonomic computing*, whose objective is to design systems able to execute themselves, adjust their behaviour in face of various circumstances, manage at best their resources and self-repair when needed.

These kind of applications are what we call *neo-computation problems*, namely: autonomic computing, pervasive computing, ubiquitous computing [12], 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. Thus, *neo-computation* systems 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;
– a dynamic and unpredictable environment;
– a functional adequacy[1] to reach in respect to the environment.

1.1 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. Looking at natural systems [3] -biological, physical, sociological-,there is a common factor among 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.

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* [2] has acquired an international renown for its works on complexity, adaptive complex systems and thus emergence. These are also the preoccupations of *Exystence* [1], the European excellence network on complex systems. More recent (Mars 2000), this network wants to promote collaboration between researchers from any field interested in it, from fundamental concepts to applications.

1.2 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 3) on one hand, and on a multi-agent approach called *AMAS* (Adaptive Multi-Agent System) [7] on the

[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".

other hand. A sub-problem (a *mathematical 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

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 [10], 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 solve 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 [7]. 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).

2.1 The Instruction-Agents and the Reorganization Process

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.

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).

2.2 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. As the programmer with his classical programming environment, the *neo-programmer* will 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. 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 *mathematical 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*) [6]. 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 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.

Whereas emergence has been studied for a long time only as a philosophical concept manipulable only as it, the self-organization field has from the very beginning tried to explore its internal mechanisms. 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).

3.1 Using Emergence in Artificial Systems

There are abundant definitions and descriptions of characteristics of emergence and self-organization in literature. To resume, 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* [5][8][11].

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 [4].

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.

3.2 The Engine for Self-organization

According to the *AMAS* theory[7],the designer provides the agents with local criterion to discern between cooperative and non-cooperative situations (NCS). 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, but the combination of all the partial functions produces the global emergent function. 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 Emergence of a Mathematical Function

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 mathematical 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), *AgentContantB*(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.

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 1) 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.

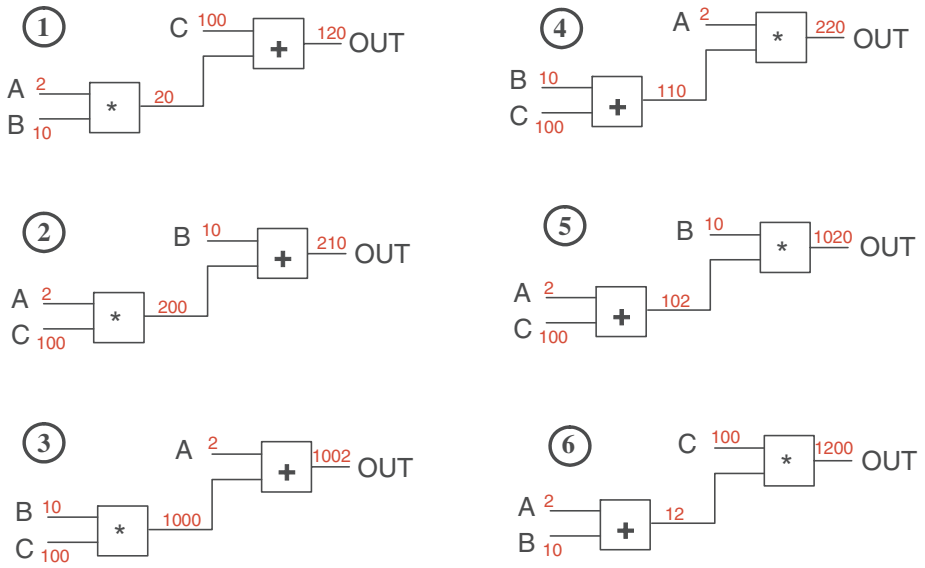

Fig. 1. The 6 different possible types of functional organizations for the mathematical example

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 [6].

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. 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 solving we implemented to be in the frame of emergence.

4.3 Results and Discussion

Results. 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 1), other NCS are produced and the system starts reorganizing toward its goal.

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 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.

A mathematical 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. Web site of exystence : the complex systems network of excellence. http://www.complexityscience.org.
2. Web site of the santa fe institute. http://www.santafe.edu.
3. S. Camazine, J.-L. Deneubourg, N. Franks, J. Sneyd, and E. Theraulaz, G.and Bonabeau. *Self-organization in biological systems*. Princeton University Press, 2002.
4. 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.
5. J. Georgé, B. Edmonds, and P. Glize. *Self-organizing adaptive multi-agent systems work*, chapter 16, pages 321–340. Kluwer Publishing, 2004.
6. 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.
7. M.-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.
8. F. Heylighen. *Encyclopedia of Life Support Systems*, chapter The Science of Self-organization and Adaptivity. EOLSS Publishers Co. Ltd, 2001.
9. P. Horn. Autonomic computing - ibm's perspective on the state of information technology. http://www.ibm.com/research/autonomic, 2001.
10. 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.
11. I. Prigogine and G. Nicolis. *Self Organization in Non-Equilibrium Systems*. J. Wiley and Sons, New York, 1977.
12. M. Weiser and J. S. Brown. Designing calm technology. *PowerGrid Journal*, 1(1), 1996.