Self-adaptive Complex Systems

Marie-Pierre Gleizes

IRIT (Institut de Recherche en Informatique de Toulouse), Toulouse University, 118 Route de Narbonne, F-31062 Toulouse Cedex 9, France
gleizes@irit.fr

Abstract. Nowadays and in the near future, the complexity of computer applications is exponentially increasing. This complexity comes from the inherent properties of such applications: the great number of their involved components, the distribution of their control and skills, the nonlinearity of their process and their increasing openness. This is also caused by the unpredictable coupling with their environment due to high dynamicity. To fulfill these requirements, systems have to adapt themselves in order to be robust and efficient. This paper will deal with self-adaptation in software systems, particularly from a multi-agent viewpoint and will focus on the Adaptive Multi-Agent Systems theory.

1 Introduction

Nowadays and in the near future, the complexity of computer applications is exponentially increasing. This complexity comes from the inherent properties of such applications: the great number of their involved components, the distribution of their control and skills, the nonlinearity of their process and their increasing openness. This is also caused by the unpredictable coupling with their environment due to high dynamicity. Complexity was previously studied in a formal way, mainly by Kurt Gödel who stated that some inherent limitations exist about completeness and consistency for formal theories including arithmetic. This means that their own validity can only be proved outside of them. Demonstration of Gödel 's incompleteness theorem introduced the computable function, formalized later in computer science (such as pi-calculus, Turing machines, recursive functions, Post's machine). These lead to main several limitations in complex artificial systems: we cannot prove in a general way that they cannot be free of bugs, and these bugs can only be detected at runtime. From this very basic result, it is possible to define some directions to design complex systems:

– A complex system must be able to self-adapt during its execution because of the dynamics but also because residual bugs are potentially included despite the design and verification phases. For the same reasons, classical learning methods cannot be sufficiently general in order to suppress these residual bugs and we must discover new approaches able to self-adapt at the micro-level without any knowledge of the global goal to achieve at the macro-level.

– Each component of the system has only to be coupled with a small number of the total amount composing the global system. This constraint comes from the previously quoted inapplicability of the global function at the micro-level, but implies the definition of micro-theories able to converge towards the desired emergent global functions.
– Designing systems in a top-down manner presupposes that the assembling of the specified components provide the desired global behaviour. Unfortunately, it is well-known that non-linearities and the multiplicity of dynamics lead to emergent phenomena at the macro-level. Consequently top-down design is fundamentally inappropriate for complex systems design.

These reasons lead us to change the perspective in order to design complex adaptive systems from satisfaction of system requirements by global and top-down activity to satisfaction of system requirements by local and bottom-up activity. Self-organisation is a way to achieve this change by allowing the design of system with emergent funtionality. To contribute to this approach, the Adaptive Multi-Agent Systems (AMAS) theory provides a guide to design self-organising systems. This theory is based on the observation that the cooperation enables to guide the agent behaviour at the micro-level, helping the agents to self-organise and to obtain adaptation at the macro-level. The modification of the interactions between the agents of the system modifies also the global function and makes the system able to adapt to changes in its environment. The interactions between agents depend on the local view these agents have and on their ability to "cooperate" with each other.

In section 2, the motivations leading to design self-adaptive complex systems are expounded. Then, section 3 presents the concepts of self-organisation and emergence. Section 4 details the AMAS theory which is studied in the SMAC[1] research group in Toulouse and has lead to Research led to a Spin off, UPETEC[2]. Then, section 5 concludes and proposes some perspectives.

2 Motivations

Nowadays, applications to design and problems to solve become more and more complex such as energy management [1], aircraft design [2], crisis management [3], maritime surveillance systems [4], ambient systems [5]... This complexity is due to a combination of aspects such as the great number of components involved in the applications, the fact that knowledge and control have to be distributed, the presence of non linear processes in the system, the fact that the system is more and more often open, its environment dynamic and the interactions unpredictable [6]. So, these properties motivate designer to realize software systems taking into account scalability, difficulty to solve problems, dynamics and under-specifications. In order to tackle the design of such complex

[1] SMAC Systèmes Multi-Agents Coopératifs `www.irit.fr/SMAC`
[2] `www.upetec.fr`

systems, self-adaptive multi-agent systems represent a promising approach providing the needed robustness and adaptation in the light of the aforementioned difficulties.

2.1 Scalability

The scalability of MAS [7] means that the system well behaves with a small numbers of agents but also with a great number of agents. A scalable system is a system which reaches scalable solutions that is to say with "reasonable" performances regardless of problem size.

The need of large-scale systems is an obvious fact in numerous domains as traffic control systems, maritime surveillance systems, simulation of biochemical reactions [8]. These domains provide strong constraints for the designer. The first one concerns the **control** which must be decentralized. Decentralization is required because it is too much difficult to get all informations about all entities in a single point without leading to a bottleneck. Decentralization enables to improve the system performances. The second point is about the **solving** which also must be distributed. It is a quite obvious consequence of the first point and this is also inevitable in systems composed of different physical agents such as cars or aircrafts in a traffic control system. The last point refers to the **accessibility of a global knowledge** about the system. Usually, because the system is too large, an agent inside the system has only a partial knowledge about the system,which is acquired from its environment. Depending on the size of the system, a global knowledge is available or not.

2.2 Problems Difficult to Solve

Several reasons can explain the difficulty for humans to solve some types of problems. The problem can present some non-linearity, some interdependencies between its parameters. The most representative classes of these problems are the multi-disciplinary, multi-objective, multi-level optimisation problems [2] and the distributed constraint-satisfaction problems [9]. The recognition of the whole problem is not possible because of its complexity. This has led to design self-organising systems presenting emergent functionalities [10], [11]. Those systems are necessarily composed of several autonomous interacting agents, plunged into an environment. In general, the global behaviour of the system emerges from the local interactions between agents. The potential of this approach is important because it simplifies the design and diminishes the design delays. But this is not so easy to do, as Van Parunak & Zambonelli [12] have claimed: "Such behaviour can also surface in undesirable ways". So, systems can reach undesirable states because the main difficulty lies in controlling the global behaviour while designing at micro-level.

2.3 Dynamics

Challenges of current systems are to take into account the dynamics. This dynamics is the result of changes that may be endogenous or exogenous to the system. Endogenous changes are generated by the system and derive, for example, from hardware failures or errors of the software behavior. Exogenous disturbances come from the system environment. To enable the system to continue to fulfill the function for which it was designed, it is necessary that it can adapt in real time. The system must also be open which means that an entity of the system (agent) may be added or removed at runtime. Disturbances from the environment of the system cause it to adapt.

2.4 High Level Expression of Requirement or under-Specified Problems

The way we usually design computational systems requires for the designer, to have some important initial knowledge. The first information a designer has to know is the exact purpose of the system. But sometimes, the main goal of the system is described at a very high level such as, for example: the system must satisfy the end-user. It is difficult to freeze this goal at the design time. Or the solution to be reached can be modified during the execution and in order to do this, the end-users have to be able to interact with the system. The system and the user have to co-construct the solution. It is the case, for example, for complex constraints problem solving in which the user can relax some constraints. The second kind of knowledge that the engineer has to know, concerns every interaction to which the system may be confronted in the future. The environment of the system can evolve and by consequences, the designer is not able to know all these situations at design time. The evolution of computer science forces us to consider that it is more and more difficult - if not impossible - not only to control accurately the activity of software with increasing complexity but also to describe completely how they work [17]. Making systems more autonomous and more adaptive is a way to simplify the task of the designer. That means that systems are able to modify their behaviour in order to achieve what they have to do at a given time.

3 Self-adaptation

As we have seen in section 2, the system to be designed must tackle complexity by using self-adaptation.

3.1 Self-adaptive Systems

In [13], according to the DARPA definition, it is said ”self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible”. This definition must be completed by the fact that the system realizes the adaptation in an autonomous way, without the designer intervention and at runtime. So, we do not have to stop the system in order to adapt it, it is able to adapt its behaviour, by itself, without stopping its excecution. This explains the prefix ”self”.

Designing these self-adaptive systems requires a radical change of perspective. Classicaly, designers satisfy the requirements by a global and top-down activity. They usually know the purpose of the system (main objective) and the interactions corpus in the future between the system and its environment. Designers must switch to the requirements satisfaction by a local and bottom-up activity. Furthermore, they do not know the purpose of the system and the body of interactions occuring in the future between the system and its environment. One of the most well-known mechanism used to enable adaptation of a system is inspired from natural systems and social animals like ants, termites... and is called self-organisation.

3.2 Self-organisation and Emergence

The works of the Agentlink Technical Forum on Self-Organisation in Multi-Agent Systems[14] have established two definitions of self-organising systems:

- **Strong self-organising systems** are systems that change their organisation without any explicit, internal or external, central control;
- **Weak self-organising systems** are systems where reorganisation occurs as a result of an internal central control or planning.

Furthermore, self-organisation implies organisation, which in turn implies some ordered structure and component behaviour. In this respect, the process of self-organisation changes the respective structure and behaviour and a new distinct organisation is self-produced [14]. Finding a solution with this kind of systems is equivalent to find the right organisation [15]. For designers, the benefits of self-organising systems are mainly due to the fact that the resolution process as a whole is not to be designed. To develop a complex system, it is sufficient to design its agents, to provide them with means to self-organise and to enable them to interact with the environment. Then, the solving process is self-constructed: it emerges from the interactions between agents [6].

The concept of self-organisation is often coupled with the concept of emergence. Emergence is the result of a collective activity and self-organisation is the means to obtain an emergent phenomenon. And it seems that emergence is a suitable context to design complex systems that cannot be controlled by a human in a centralised way. We commonly agree with the fact that an emergent phenomenon must be observable. From an observer point of view, we assume that if one can observe the content of the entities of a system and if one can observe at the system level a behaviour that cannot be reduced to the behaviour of the entities, the global behaviour can be qualified as emergent. In other words, we can say that a human cannot determine the global behaviour of the system only by looking at the agent behaviour. We can also qualify a phenomenon as emergent if we need different terms, vocabularies to explain the micro and the macro levels. This leads to give the following operational definition of emergence in artificial systems, based on three points: what we want to be emergent (subject), at what condition it is emergent and how we can use it (method) [16].

1. **Subject.** The goal of a computational system is to realise an adequate function, judged by a relevant user. This ”function” can be for instance a behaviour, a pattern, a property (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 a coherent and relevant function.
3. **Method.** The mechanisms which allow the changes are specified by self-organisation rules, providing autonomous guidance to the components' behaviour without any explicit knowledge about the collective function nor how to reach it.

3.3 Self-organisation / Re-organization

Two well-known communities focus on the design of adaptive and robust systems: SASO(Self-Adaptive and Self-Organising systems) and COIN (Coordination, Organisation, Institutions and Norms in agent systems). The SASO community studies self-adaptation and self-organisation. In a self-organising multi-agent systems context, the designer of such systems first focuses on agent local behaviours and peer-to-peer interactions. The organisation is the result of the collective emergent behaviour due to how agents act their individual behaviours and interact in a common shared and dynamic environment. The designer does not put any global knowledge about the organisation inside the agents.

In the COIN community, the designer provides on one hand, the entire organisation and coordination patterns, and on the other hand, the agents local behaviours . At runtime, the agents may consider the constraints imposed by the defined organisation as compulsory or possible guidelines for the coordination of their local behaviours. The organisation exists at design time. The designer puts this knowledge inside the system and allows the agent access to this information.

Both communities are interested in adaptation and self-adaptation and use the organisation concept but the hypothesis for designing systems are different. Organisation is a first-abstract class in the design of COIN systems, required at design time. In self-organising systems, the organisation is the result of the collective behaviour.

3.4 Self-organisation Mechanisms

Currently, a lot of mechanisms of self-organisation are implemented in artificial systems [11] and cannot be all detailed here. The older are nature-inspired mechanism and copy the activity of social animals such as: foraging, nest building, sorting, web weaving... The most well-known technique is the **stigmergy** mechanism. It has been widely used and was first observed in societies of social insects by Grassé and can be summarised as "*the work excites the workers*" [17]. Agents leave information in the environment which can be perceived by the others. This information, usually evaporates after a given time. This mechanism allows task coordination and regulation within a group, using only indirect interactions and without any central control. There is no method to develop this technique and the primary difficulty is to adjust the different parameters such as the speed of evaporation or the amount of information dropped. Because the solution must be represented in the environment, the final goal of the system guides the design phase. Some mechanisms are inspired from physics such as the gradient fields. They are based on attraction and repulsion behaviours and show self-organised behaviour [18]. We can also find mechanisms imitating human behaviour such as gossip. It enables to spread information and to create evolving organisations [19].

In all these works, researchers have applied a mechanism observed or not in the nature, a theory to an application. Sometimes the phenomenon observed by simulating the system is interesting such as web weaving. But it is difficult to find a real application which can be realized by the system. Concerning web weaving, researchers have applied it to regions detection but they do not propose another application. What we try to do in my research group is to constantly go from applications to theory but also from the theory to applications. We realize specific applications by highlighting generic behaviours and then, we reuse these behaviours by improving them to design other applications. Applications enrich theory and theory enables to develop new applications. This theory: the Adaptive Multi-Agent Systems theory [20],[21],[16], [6] is presented in the following sections.

4 AMAS Theory

The first aim of the AMAS theory is to design MAS having a coherent collective activity that achieves the right task. We name this property "functional adequacy" and we proved the following theorem: "For any functionally adequate system, there is at least a cooperative interior medium system which fulfills an equivalent function in the same environment". Therefore, we focused on the design of cooperative interior medium systems in which agents are in cooperative interactions. The specificity of the theory resides in the fact that we do not code the global function of the system within the agents. Agents have only a partial knowledge. The global function of this system emerges from the collective behavior of the different agents composing it. Each agent possesses the ability of self-organisation i.e. the capacity to locally rearrange its interactions with others depending on the individual task it has to solve. Changing the interactions between agents can indeed lead to a change at the global level. This induces the modification of the global function. This capacity of self-organisation enables to change the global function without coding this modification at the upper level of the system. An intuitive example is the realisation of an elementary mathematical function. Let five agents be *, +, 2, 100, 5, if the organisation is the following: 2+(5*100) the result is 502 but if the organisation was (2+5)*100, the result would be 700. With this simple example, we can see that a change inside an organisation, changes the results provided by it. Self-organization in AMAS is based on the capacity an agent possesses to be locally "cooperative".

These agents, called **cooperative agents** are composed of five parts contributing to their behavior: skills (what the agent is able to do), representations of the world (the knowledge it has about itself, about the others or about its environment), an interaction language (to communicate with others or with its environment), aptitudes (the capacities it possesses to reason on its knowledge) and a social attitude led by what we call cooperation.

4.1 Definition of Cooperation

The basic definition of the cooperation between two agents is that an agent helps the other to execute a task. For example, two agents are needed to carry a very heavy object and they must help each other to do it. In Artificial Intelligence [22] cooperation means that, if two agents have two different goals then the fact that an agent can reach its goal, does not prevent the other to reach its own one. In the AMAS theory, cooperation defines more an attitude, a behaviour that an agent has to follow.

In the AMAS theory, an agent is benevolent, sincere, willing, fair and implements reciprocity. Benevolence is different from altruistism and means that for a limited duration, an agent can leave its individual goal to help another agent or to adopt the goal of another one. An agent is sincere if it never lies to other agents. A willing agent is an agent which tries to satisfy a request if it is coherent with its own skills and the current state of the world, and if no damage results from the action, either to the acting agent or to another. If there is a resulting damage, refer to property four. A fair agent always tries to satisfy, when it is possible, agents with the highest level of difficulty for reaching their goals. Reciprocity is the fact that each agent of the same society knows that itself and the others verify these four main properties.

4.2 Cooperative Agent Behaviour

A cooperative agent aims at always being in a cooperative state that is in cooperative interactions with its environement. However, because of the dynamic nature of the environment of the system, as well as the dynamics of the interactions between agents, an agent can be in a non cooperative state or can be at the origin of cooperation failures. We call these states: "Non Cooperative Situations" (NCS). More precisely an agent can detect NCS at three different steps during its lifecyle:

- when a signal perceived from its environment is not understood and not read without ambiguity;
- when the information perceived is not useful for the agent's reasoning;
- when concluding results lead to act in a useless way in the environment.

The general algorithm followed by a cooperative agent (see 1) consists in executing what the agent has been created for (called its Nominal behaviour), if it does not detect NCS. But if a cooperation problem occurs, it can realize one or more of the three following behaviours: tuning, reorganisation and evolution. For tuning, an agent changes the value of some of its internal parameters. For reorganising, an agent modifies the way it interacts with its neighbours by: adding or removing a neighbour, modifying the trust it has in a neighbour. Concerning the evolution, the agent self-removes or creates another agent (a replication of it for example). Note that these behaviours lead to self-organisation.

```
Begin while The agent is alive do
    if The agent does not detect any Non Cooperative Situation then
        Execute the Nominal Behaviour;
    else
        The agent tries to:
        Adjust its internal parameters: Tuning;
        and/or Change its interactions: Reorganization;
        and/or Add or remove an agent: Evolution;
    end
end
```

Algorithm 1. Cooperative agent's behaviour

In the algorithm, we can note that the detection of NCS infuences the agent behaviour. As a consequence, a central point in the AMAS theory has been the definition of NCS. At this step of our work, we have registered seven NCS [23]:

- **Incomprehension** is related to the interpretation of the messages and informs that the agent is not able to extract any understandable information from the received message.
- **Ambiguity** informs the agent that different interpretations are possible, and therefore, an accurate representation update is not possible. That can be due for instance to missing information.
- **Incompetence** is detected when the agent does not have the competence to process the received information such as answering an agent request.
- **Unproductivity** is detected when the agent has accurately interpreted the received information but cannot use it to produce any useful information for itself because it already has this information, it is of no interest for it or the received information is incomplete.
- **Conflict** is detected either when considering the list of possible future actions or when detecting a conflict in the environment. In the first case, among the list of possible actions some are conflicting and such actions cannot be performed by the agent at the same time (i.e. lack of resources). In the second case, the conflict can either be due to a previous action performed by the agent or another agent, or a change in the environment not related to the agent activity. This situation is also detected when the agent considers

that modifying the environment can prevent other agents from reaching their goals.
- **Concurrence** concerns the interactions between the agent and its environment. It is detected when among the list of possible actions, some can put the agent in concurrence or competition with other agents.
- **Uselessness** is detected when the agent considers itself not useful for the system or its environment. This can be due to a lack of information or unused knowledge.

Depending on the application, only some of these NCS are relevant. For example, in an ant simulation, the notion of incomprehension is not necessary because an ant can always understand what it perceives (food, pheromone, ant, obstacle).

4.3 Mechanisms for Implementing Cooperation

Cooperation is considered as an attitude which guides the agent behaviour in using only a partial knowledge: the agent's knowledge and its local perceptions. The cooperation is implemented inside an agent with the four following mechanisms:

- spontaneously communicate,
- anticipate cooperation failures,
- repair cooperation failures,
- act for helping its worst neighbours.

Notice that depending on the application, not all these mechanisms must be implemented by the designer inside the agent.

Spontaneous Communication. Spontaneous communication consists in communicating to an agent an information not requested by it. An agent sends a piece of information if it thinks that this information can be useful to another agent. An example of this action has been implemented in the simulation of foraging ants [24]. In this simulation, the environment is composed of the nest, some obstacles, pheromone, pieces of food and ants. The pheromone self-evaporates during time and can be accumulated when several ants drop pheromone at the same place. The foraging ants are the cooperative agents of the multi-agent system. Their behaviour is copied from natural ants and consists first in exploring the environment. When it encounters an obstacle, it avoids it. When it encounters food, it can harvest it. When it is loaded, it goes back to nest in dropping a given quantity of pheromone on the ground. By consequence, tracks of pheromone appear in the environment. During its exploration, an ant is attracted by pheromone and leads to follow pheromone track. This behaviour implies a reinforcement of the existing tracks. In the following situation called "come back to the nest": when an ant is loaded and comes back to the nest, it puts pheromone on the ground to mark the location. The spontaneous communication is implemented as follows: in the "come back to the nest" situation

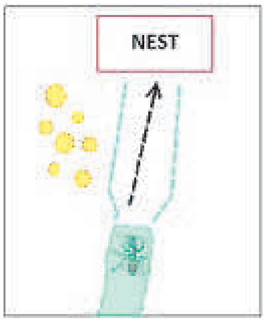

Fig. 1. Spontaneous communication of an ant

if in addition the ant perceives new pieces of food, it drops a higher quantity of pheromone on the ground (see figure 1). This is done to provide a better information about the environment to other ants.

Anticipation of Non Cooperative Situations. If it is possible, an agent can try to anticipate NCS. The designer has to implement the detection of possible NCS due to the future action of the agent. If an agent can know in advance that its action will lead to a NCS, it has to avoid it. For example, an agent can anticipate an **Unproductivity** NCS by informing other agents about the update of its profile such as the agents main interest or the information produced by the agent and judged helpful for others. The NCS anticipation can be illustrated in

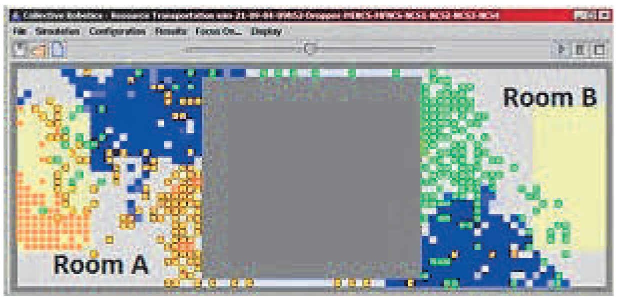

Fig. 2. Carrier robots

the carrier robots application [25]. In this application, see figure 2, there are two rooms A and B, separated by two narrow corridors (where two robots cannot cross). The robots have to take a box in the room A and to drop it in the room B. They have only local perceptions (only adjacent cases). For each robot, we store the ten last locations where the robot has encountered problems (most of the time a location where it cannot move because something is in front of it). These locations are in blue in the figure 2. There are a lot because all locations marked by all robots are visualised on this figure. Of course for one robot there are no more than ten blue locations. Therefore thanks to this memory a robot can anticipate that there is a robot coming in the opposite diretion in front of it, and choose another direction to avoid the conflict.

Treatment of Non Cooperative Situations. If an agent detects NCS, it must act to repair them. By consequence, the designer has to provide for each NCS an handler which will be executed by the agent to come back in a cooperative state. This handler is application-dependent. For example, if an **Incomprehension** NCS is detected, to solve this NCS, the agent can for instance ask the sender to modify its message, or ask other agents that may understand it for a translation/decryption.

The NCS treatment can be illustrated in the carrier robots application see figure 2. A conflict occurs when inside the corridor, one robot is in front of another robot moving in the opposite direction. In this case, if it is possible, the agent must move to its sides (left or right). If it cannot move laterally, two other solutions are opened. If the other robot has an antagonist goal, the robot which is the most far from its goal will move backward to free the way for the robot which is the closest to its goal. Robots can evaluate which is the most distant since they know their goals and the associated zones.

Action to Help Its Worst Neighbour. Each agent can measure the degree of difficulty it has to reach its individual goal. This measure is called criticality. When an agent receives a request from its neighbours, the request can be provided with the criticality of the sender agent. So a cooperative attitude is to try to help its neighbours which have the most difficulties to reach their goals. In fact, in doing this the satisfaction of all the agents in the system tends to be balanced. No agent is very satisfied and no agent is not satisfied at all.

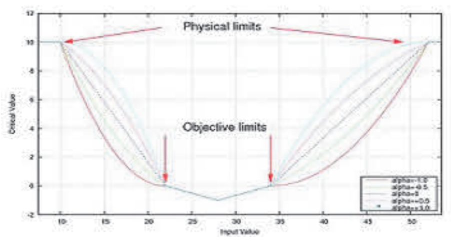

Fig. 3. Criticality in the MASCODE system

The criticality notion can be showed in the MASCODE system [2]. MASCODE is an aircraft design system which realizes multi-disicplinary and multi-objective optimisation. When designing a new aircraft, a designer defines his objective parameters, the main constraints on the design and the main expected performances. Each of these parameters are agentified. The criticality of an agent provides information on the fact that the value of the parameter (it represents) found by the agent is inside or outside the validity interval (see figure 3). With the curve of figure provided to the agent, this latter is able to compute its criticality. Within the objective limits, the agent is completely satisfied and it is not critical. Outside its physical limits, it is not at all satisfied and its criticality is equal to the maximum value.

4.4 Designing Self-adaptive MAS

To facilitate the design of systems based on the AMAS theory, we have proposed a methodology called ADELFE [26] and a framework called MAY[3]. These two tools will not be described here, only the main steps the designer has to follow are summarized. First of all the designer has to determine the agents and their nominal behaviour. He has also to verify which mechanism is needed and relevant i.e. he has to decide if spontaneous communication is needed. Then, he has to find for every type of agents, which NCS among the seven NCS (described in section 4.2) an agent can encounter. For every possible NCS, he has to provide a handler to treat this NCS and for every anticipated NCS, to provide a handler to avoid it. At the end, he has to explore the utility of using or not criticality notion.

All these steps are more or less simple, the expertise acquired in building AMAS systems is of course very useful to design a new system. But the main difficulty is to think "local". Because the designer knows what the collective has to do, he tends to put this knowledge inside the agents. This is the wrong way to design adaptive complex systems. Because of the difficulty to realize the global task or at the deployment phase with a great number of agents, this global knowledge is no more accessible.

5 Conclusion

Self-adaptive complex systems are relevant systems to cope with scalability, dynamics, difficulty to solve a problem, and under-specification. The AMAS theory, presented in this paper, is our contribution to design a class of self-adaptive systems: the self-adaptive multi-agent systems. This theory was and is also currently applied in numerous research projects but also in industrial ones: maritime surveillance, aircraft electric harness optimisation, energy management,... Future systems will be composed of systems and will be system of systems which will not be designed by the same designer. In this context, interoperability and openess will be the main future research challenges. The large scale of systems will require to find new means to evaluate and validate them. The context in which

[3] `www.irit.fr/MAY`

the system will be deployed will not be known in advance. The designer will have to tackle unanticipated adaptation. This means that he will not precisely know this context at design time. Self-adaptive complex systems should be improved to answer to these fascinating and open challenges.

Acknowledgements. I want to thank all members of the SMAC team: permanent researchers, students supervised in the past and ongoing PhD who contributed to the work presented in this paper.

References

1. Lagorse, J., Paire, D., Miraou, A.: A multi-agent system for energy management of distributed power sources. Renewable Energy 35(1), 174–182 (2010)
2. Welcomme, J.B., Gleizes, M.P., Redon, R.: Adaptive multi-agent systems for multidisciplinary design optimisation. In: 16th International Conference on Engineering Design (ICED 2007), Paris, The Design Society (2007)
3. Lacouture, J., Gascueña, J.M., Gleizes, M.-P., Glize, P., Garijo, F.J., Fernández-Caballero, A.: ROSACE: Agent-Based Systems for Dynamic Task Allocation in Crisis Management. In: Demazeau, Y., Müller, J.P., Rodríguez, J.M.C., Pérez, J.B. (eds.) Advances on PAAMS. AISC, vol. 155, pp. 255–260. Springer, Heidelberg (2012), `http://www.springerlink.com`
4. Mano, J.-P., Georgé, J.-P., Gleizes, M.-P.: Adaptive Multi-agent System for Multi-sensor Maritime Surveillance. In: Demazeau, Y., Dignum, F., Corchado, J.M., Pérez, J.B. (eds.) Advances in PAAMS. AISC, vol. 70, pp. 285–290. Springer, Heidelberg (2010)
5. Georgé, J.P., Camps, V., Gleizes, M.P., Glize, P.: Ambient Intelligence as a Never-Ending Self-Organizing Process: Analysis and Experiments. In: ASAMi 2007 (2007)
6. Gleizes, M.-P., Camps, V., Georgé, J.-P., Capera, D.: Engineering Systems Which Generate Emergent Functionalities. In: Weyns, D., Brueckner, S.A., Demazeau, Y. (eds.) EEMMAS 2007. LNCS (LNAI), vol. 5049, pp. 58–75. Springer, Heidelberg (2008)
7. Turner, P.J., Jennings, N.R.: Improving the Scalability of Multi-agent Systems. In: Wagner, T.A., Rana, O.F. (eds.) Infrastructure for Agents 2000. LNCS (LNAI), vol. 1887, pp. 246–262. Springer, Heidelberg (2001)
8. Videau, S., Bernon, C., Glize, P.: Towards Controlling Bioprocesses: A Self-adaptive Multi-agent Approach. Journal of Biological Physics and Chemistry 10(1), 24–32 (2010)
9. Liu, J., Jing, H., Tang, Y.Y.: Multi-agent Oriented Constraint Satisfaction. Artificial Intelligence 136(1), 101–144 (2002)
10. Di Marzo Serugendo, G., Gleizes, M.P., Karageorgos, A.: Self-Organization in Multi-Agent Systems. The Knowledge Engineering Review 20(2), 165–189 (2005)
11. Di Marzo Serugendo, G., Gleizes, M.P., Karageorgos, A. (eds.): Self-organising Software - From Natural to Artificial Adaptation. Natural Computing Series. Springer (Octobre 2011), `http://www.springerlink.com`
12. Zambonelli, F., Van Dyke Parunak, H.: Signs of a Revolution in Computer Science and Software Engineering. In: Petta, P., Tolksdorf, R., Zambonelli, F. (eds.) ESAW 2002. LNCS (LNAI), vol. 2577, pp. 13–28. Springer, Heidelberg (2003)

13. Robertson, P., Laddaga, R., Shrobe, H.: Introduction: The First International Workshop on Self-Adaptive Software. In: Robertson, P., Shrobe, H.E., Laddaga, R. (eds.) IWSAS 2000. LNCS, vol. 1936, pp. 1–10. Springer, Heidelberg (2001)
14. Di Marzo-Serugendo, G., Gleizes, M.P., Karageorgos, A.: Self-Organisation and Emergence in MAS: An Overview. Informatica 30(1), 45–54 (2005)
15. Georgé, J.P., Picard, G., Gleizes, M.P., Glize, P.: Living Design for Open Computational Systems. In: International Workshop on Theory and Practice of Open Computational Systems (TAPOCS) at IEEE 12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2003), Linz, Austria, Juin 09-Juin 11, pp. 389–394. IEEE Computer Society (2003), `http://www.computer.org`
16. Capera, D., Georgé, J.P., Gleizes, M.P., Glize, P.: Emergence of Organisations, Emergence of Functions. In: AISB 2003 Symposium on Adaptive Agents and Multi-Agent Systems, pp. 103–108. University of Wales, Aberystwyth, Society for the Study of Artificial Intelligence and the Simulation of Behaviour (2003)
17. Grassé, P.: La reconstruction du nid et les interactions inter-individuelles chez les bellicositermes natalenis et cubitermes sp. La théorie de la stigmergie: essai d'interprétation des termites constructeurs. Insectes Sociaux 6, 41–83 (1959)
18. Mamei, M., Zambonelli, F.: Field-based Coordination for Pervasive Multiagent Systems. Springer Series on Agent Technology (2006)
19. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A., van Steen, M.: Gossip-based peer sampling. ACM Transactions on Computer Systems 25(3) (2007)
20. Camps, V., Gleizes, M.P., Glize, P.: A self-organization process based on cooperation theory for adaptive artificial systems. In: 1st International Conference on Philosophy and Computer Science "Processes of evolution in real and Virtual Systems", Krakow, Poland (1998)
21. Gleizes, M.P., Camps, V., Glize, P.: A Theory of Emergent Computation Based on Cooperative Self-Oganization for Adaptive Artificial Systems. In: 4th European Congress of Systems Science (1999)
22. Ferber, J.: Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. Addison Wesley (1999)
23. Kaddoum, E.: Optimisation sous contraintes de problèmes distribués par auto-organisation coopérative. Thèse de doctorat, Université de Toulouse, Toulouse, France (Novembre 2011)
24. Topin, X., Fourcassié, V., Gleizes, M.P., Theraulaz, G., Régis, C.: Theories and experiments on emergent behaviour: From natural to artificial systems and back. In: European Conference on Cognitive Science, Siena (Octobre 1999)
25. Picard, G., Gleizes, M.P.: Cooperative Self-Organization to Design Robust and Adaptive Collectives. In: 2nd International Conference on Informatics in Control, Automation and Robotics (ICINCO 2005), Barcelona, Spain, September 14-17, vol. I, pp. 236–241. INSTICC Press (2005)
26. Rougemaille, S., Migeon, F., Maurel, C., Gleizes, M.P.: Model Driven Engineering for Designing Adaptive Multi-Agent Systems. In: European Workshop on Multi-Agent Systems (EUMAS), Hammamet, Décembre 13-Décembre 14. Ecole Nationale des Sciences de l'Informatique, ENSI, Tunisie (2007) (electronic medium)