Engineering Systems which Generate Emergent Functionalities

Marie-Pierre Gleizes, Valérie Camps, Jean-Pierre Georgé, Davy Capera

Engineering Systems Which Generate Emergent Functionalities

Marie-Pierre Gleizes, Valérie Camps,
Jean-Pierre Georgé, and Davy Capera

Université Paul Sabatier, IRIT,
118 route de Narbonne, 31062 Toulouse Cedex 9, France
{gleizes,camps,george,capera}@irit.fr
http://www.irit.fr/SMAC

Abstract. Complexity of near future and even nowadays applications is exponentially increasing. In order to tackle the design of such complex systems, being able to engineer self-organising systems is a promising approach. This way, the whole system will autonomously changes its behaviour as its parts locally reorganise themselves, always providing an adapted function. This paper proposes to focus on engineering such systems generating emergent functionalities. We will first define two important concepts to take into account in such a context: Emergence and Self-Organisation. Building on these two concepts, we will highlight three main challenges researchers have to cope with: *(i)* how to control the system at the macro level by only focusing on the design of agents at the micro level, *(ii)* what kind of tools, models and guides are needed to develop such systems in order to help designers and *(iii)* how validation of such systems can be achieved? Each of these three challenges will be explained and positioned in regard to the main existing approaches. Our solutions combining emergence and self-organisation will be expounded for each challenge.

Keywords: Complex Systems, Engineering, Emergence, Multi-Agent System, Self-Organisation.

1 Context, Definitions and the Three Challenges

Complexity of near future and even nowadays applications is exponentially increasing. This is due to a combination of aspects such as the great number of components taking part 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. In order to tackle the design of such complex systems, being able to engineer self-organising systems is a promising approach providing the needed robustness and adaptation in the light of the aforementioned difficulties.

1.1 Understanding and Designing Complex Artificial Systems

The multidisciplinary community ONCE-CS (Open Network of Centres of Excellence in Complex Systems), clearly states the current interests in Complexity: "*the new Science of Complex System addresses the need to master the increasing complexity we see in natural and social systems. Examples include the human and new treatments for disease, managing the Internet, public administration, and business. This new science will revolutionise our world, causing irresistible changes*". In computer science, different kinds of techniques have been developed to tackle complexity such as those based on heuristics and metaheuristics [1], those based on learning such as genetic algorithm or neural network [2,3] and those based on self-organisation processes [4,5,6]. Since a Multi-Agent System (MAS) is defined as a macro-system composed of autonomous agents which pursue individual objectives and which interact in a common environment to solve a common task, it can be viewed as a paradigm to design complex applications.

To overcome difficulties coming from the openness and the dynamic of the environment, the system must be adaptive. Most natural systems have the ability to adapt themselves to a changing environment, such as the ability of the body to adapt its internal temperature when the temperature outside changes. It is well known that the process enabling these phenomena is self-organisation, defined by Bonabeau et al. as: *"a set of dynamical interactions whereby structures appear at the global level of a system from interactions among its lower-level components [...] The rules specifying the interactions are executed on the basis of purely local information, without reference to the global pattern"* [7].

1.2 Defining Self-organisation and Emergence for Artificial Systems

Self-organisation is a paradigm more and more used in MAS [8] and a definition with an artificial systems point of view has been provided by the European working group TFG SO (TFG SO[1] of Agentlink III) [9]:

Definition 1. *Self-organisation is the mechanism or the process enabling a system to change its organisation without explicit external command during its execution time.*

In general, the environment plays a fundamental role in the self-organisation process in constraining the system behaviour. It provides events which disturbs the system and leads the system to change its behaviour in self-organising. But some artificial systems can self-organise without interaction with the environment. In this case, when the system becomes stable it cannot evolve more.

The concept of self-organisation is often coupled with the concept of emergence [10]. 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.

[1] TFG SO: Technical Forum Group on Self-Organisation in MAS, see `http://www.irit.fr/TFGSO`.

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[2]. 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) [11,12].

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.

1.3 The Three Challenges for Engineering Systems Which Generate Emergent Functionalities

Designers of complex systems have been taking inspiration from natural systems in which complex structures or behaviours appear at the global level of a system from interactions among its lower-level components. The phenomenon observed at the macro-level emerges by self-organisation of the micro-level components making up the system. From an engineering point of view, the potential of this approach is important because it simplifies the design and diminishes the design delays. To develop a complex system, it is sufficient to design its components (called agents) which are less complex, to provide them with means to self-organise through local interactions and to enable them to interact with parts of the environment. But this is not so easy to do, as Parunak & Zambonelli [13] have claimed: "Such behaviour can also surface in undesirable ways". So, systems can reach undesirable states because the main difficulty lies in controlling global behaviour while designing at micro-level.

In our point of view, there are three main challenges to overcome to design self-organising applications. The first consists in answering the question: "how to control the emergence" or in others terms "how to control the system behaviour at the macro level by only focusing on the design of agents at the micro level?"

[2] This criteria has been highlighted in the working group TFG SO.

in order to avoid harmful global phenomena. The second challenge is to provide tools, models and guides to develop such systems. Because the goal of engineering self-organising systems is to deliver systems with a global behaviour which meets the requirements or realizes the desired function, the third challenge is about how to validate these systems. The aim of this paper is to briefly present these challenges in regard to the main existing approaches and to expound our ideas and solutions to address them.

In this paper, we first define the two important concepts to take into account in such a context: Emergence and Self-Organisation (Section 2). Then, the three challenges will be respectively explained in section 3,4 and 5, positioned in regard to the main existing approaches and finally our approach will be expounded. The paper ends by stating which research axes have to be pursued.

2 Challenge 1: An Emergence-Based Theory for the Designer of Complex Systems

Designing such MAS requires to find rules to make the system achieves the required collective behaviour, that is "functions that are useful to the system's stakeholders" [14], "the required macroscopic behaviour" [15], "a functionally adequate function" [11] ,... How does this produce a complex system with the right behaviour at the global level? The environment plays here its key role by constraining the system, and the system needs to be able to adapt to these constraints. There is an apparent antinomic situation in the idea of engineering applications with emergent functionalities. On one hand, emergent behaviour is a behaviour which occurs and in a certain manner cannot be under control. On the other hand, a software designer wants the system he is building to achieve a desired function. So, we can conclude saying that we want to *control the emergent behaviour* of the systems. The solution is then to better understand relations between micro and macro levels and to build a system able to self-adapt to environmental dynamics.

2.1 Some Mechanisms to Engineer Self-organising Applications

Currently, the objective of most researchers in self-organising MAS is to find relevant mechanisms to guide the agent behaviour at the micro level, helping the agents to self-organise and to obtain at the macro level, the behaviour of the system the designer expects. But the previous definition framework needs to be carefully instantiated with specific techniques enabling this self-organisation while allowing emergent functionalities to appear. Usual techniques are based on stigmergy, cooperation, gossip, natural selection, attraction and repulsion, potential fields, social relationships, trust...

One of the first kind of artificial systems related to self-organisation is based on the metaphor that only the better adapted individuals survive. In evolutionary computation and genetic algorithms [16,17], the system finds a solution in a huge state space in converging towards similar individuals which represent the

solution. They are able to learn and adapt because the population evolves under the pressure of a specific function. Designer have to face two difficulties: on one hand, to give a well suited problem representation in terms of individuals and genes, and on the other hand to provide, in addition to mutation and cross-over operators, an efficient fitness function used by the individuals. This fundamental function is in general established from global knowledge about the solution the designer wants to achieve.

Neural networks [18,19] are usually composed of several layers: the entry, output and the hidden ones. Each layer has several neurons connected by weighted links. They are able to change the organisation between neurons of two consecutive layers during the learning phase by changing the weights. In general, it is difficult to find the right number of hidden levels and the number of neurons per level. The function used to update the weights of the links is dependent of the solution the system has to reach. Moreover, the learning corpus is not so easy to choose and requires a habit from the designer. The evolution of the system can be viewed as the self-organisation of the neurons, in particular in Kohonen maps [20].

Multi-agent systems are one of the most representatives among artificial systems dealing with *complexity* and *distribution* [21,22]. Self-organisation is a way to simplify the design of these systems in having a bottom up approach. Three kinds of inspirations are used to design these self-organising systems: the biologic and natural one [23], the social one [24], and the artificial one [25]. The mechanisms based on biologic approaches are closer to the work presented in this paper.

The stigmergy mechanism 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*" [26]. Agents leave information in the environment which can be perceived by the others. In general, this information evaporates after a given time. This mechanism allows task coordination and regulation within a group, using only indirect interactions and without 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. It is quite obvious that it cannot be applied if agents cannot act directly on an environment.

2.2 Adapt the System by Its Parts

In our approach, we consider that each part P_i of a system S achieves a partial function f_{P_i} of the global function f_S (cf. figure 1). f_S is the result of the combination of the partial functions f_{P_i}, noted by the operator "$\circ$". The combination being determined by the current organisation of the parts, we can deduce $f_S = f_{P_1} \circ f_{P_2} \circ ... \circ f_{P_n}$. As generally $f_{P_1} \circ f_{P_2} \neq f_{P_2} \circ f_{P_1}$, by transforming the organisation, the combination of the partial functions is changed and therefore the global function f_S changes. So, enabling a MAS to self-organise consists in

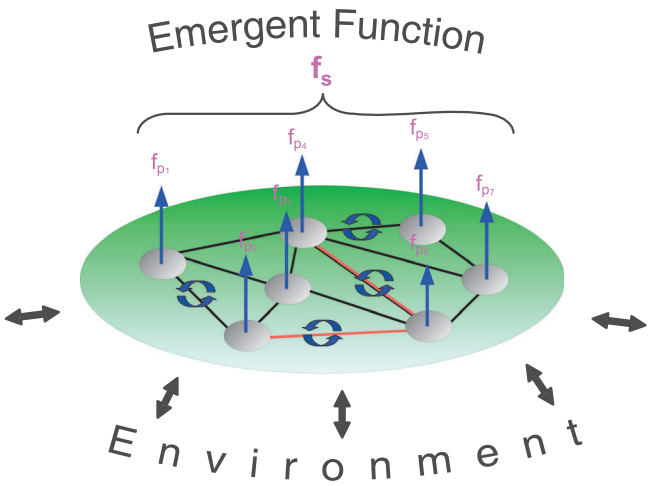

Fig. 1. Adaptation: changing the function of the system by changing its organisation

enabling the agent to change inside the organisation. The global function realized is the result of the organisation between agents in the system. This reorganisation technique can be extended with two other techniques, we are currently working on: *self-tuning* (parts can modify the parameters defining their behaviour) and *self-evolution* (parts can appear and disappear when needed). To ensure that the system will generate emergent behaviour, according to the definition of the emergence and to be able to *control* this emergence, it is necessary to provide to the agents a local criterion which enables them to self-organise. This requires both a theoretical and engineering framework.

The cooperation is the engine of the self-organisation and the heart of our bottom-up method. Cooperation is classically defined by the fact that two agents work together if they need to share resources or competences [27]. We add to this definition, the fact that an agent locally tries on one hand, to anticipate problems and on the other hand to detect cooperation failures called Non Cooperative Situations (*NCS*) and try to repair these NCS [28]. To anticipate NCS, the agent always chooses the actions which disturb the less other agents it knows. In others words, the agents, by trying to always have a cooperative attitude, act by reorganising their acquaintances and interactions with the others agents.

2.3 Controlling Emergence: The Theorem of Functional Adequacy

Cooperation was extensively studied in computer science by Axelrod [29] and Huberman [30] for instance. "*Everybody will agree that cooperation is in general advantageous for the group of cooperators as a whole, even though it may curb some individual's freedom*" [31]. In order to show how cooperation improves artificial complex systems design, we have developed the *AMAS* (Adaptive Multi-Agent System) [32,11] theory which is based upon the following theorem. This

theorem describes the relation between cooperation in a system and the collective result which is *"functionally adequate*[3] *"*.

Theorem 1. *For any functionally adequate system, there exists at least one cooperative internal medium system that fulfils an equivalent function in the same environment.*

Definition 2. *A cooperative internal medium system is a system where no* NCS *exist.*

Definition 3. *An agent is in a (*NCS*) when:*

($\neg c_{per}$) *a perceived signal is not understood or is ambiguous;*
($\neg c_{dec}$) *perceived information does not produce any new decision;*
($\neg c_{act}$) *the consequences of its actions are not useful to others.*

The objective is to design systems that do the best they can when they encounter difficulties called NCS. The designer has to describe not only what an agent has to do in order to achieve its goal but also which locally detected situations must be avoided and when they are detected how to suppress them (in the same manner that exceptions are treated in classical programs).

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 [32]:

- The system is cooperative and functionally adequate to the constraints of its environment. Its parts do not 'know' the global function the system has to achieve via adaptation (thus enabling emergent functionalities).
- 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.

2.4 Architecture and Functioning of an *AMAS* Agent

A cooperative agent in the *AMAS* theory has the four following characteristics. First, an agent is autonomous. Secondly, an agent is unaware of the global function of the system; this global function emerges (from the agent level towards

[3] "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".

the multi-agent level). Thirdly, an agent can detect NCSs and acts to return in a cooperative state. And finally, a cooperative agent is not altruistic (it does not always seeks to help the other agents), but benevolent (it seeks to achieve its goal while being cooperative).

Cooperative agents are equipped with several modules representing a partition of their "physical", "cognitive" or "social" capacities. Each module represents a specific resource for the agent during its "perceive-decide-act" life cycle. The first four modules are quite classical in an agent model: *Interaction Module* (in fact composed of *Perception Module* and *Action Module*), *Skill Module*, *Representation Module* and *Aptitude Module*. The novelty comes from the *Cooperation Module* which contains local rules to solve NCS. All the cooperative attitudes of agents are implemented in this module: it must provide an action for a given state of skills, representations and perceptions, *if the agent is in a NCS*. Therefore, cooperative agents must possess rules to detect NCS. For each NCS detection rule, the Cooperation Module associates one or several actions to process to avoid or to solve the current NCS. During the perception phase of the agents' life cycle, the Perception Modules updates the values of the sensors. These data directly imply changes in the Skill and Representation Modules. Once the knowledge updated, the decision phase must result on an action choice. During this phase, the Aptitude Module computes from knowledge and proposes action(s) or not. In the same manner, the Cooperation Module detects if the agent is in a NCS or not. In the former case, the Cooperation Module proposes an action that subsumes the proposed action by the Aptitude Module. In the latter case, the only action[4] proposed by the Aptitude Module is chosen. Once an action is chosen during the action phase, the agent acts by activating its effectors or changing its knowledge.

According to the *AMAS* theory, agents have to be able to detect when they are in a NCS and how they can act to come back in a cooperative situation. Agents also always try to stay in a cooperative situation and so the whole system converges to a cooperative state within and with its environment. This leads – according to the theorem of functional adequacy (theorem 1) – to an adequate system.

Thus, this describes the typical decision process of a generic *AMAS* agent. But the NCS and the actions which could be applied to solve them are not generic: designers have to write their own specific NCS set and related actions for each kind of agent they wish the system to contain.

3 Challenge 2: A Method and Tools for the Designer of Complex Systems

The first and obvious problem software designers encounter when trying to engineer complex systems lies of course in their nature: complexity. How can we build something we do not even fully understand? Since the years 2000, agent

[4] There is only one action possible, otherwise an NCS is detected.

oriented methodology field is in full rise; numerous new methodologies devoted to particular problems appeared [33], but very few of them are devoted to design multi-agent systems generating emergent functionalities.

3.1 Existing Works for Engineering Self-organising Multi-agent Systems

Van Parunak and Bruckner propose a design guide for swarming systems engineering [14] consisting of ten design principles: the four first are derived from couples processes, the three next are derived from autocatalysis and the three last are derived from functional adjustment. Even if swarming systems have demonstrated their effectiveness as an alternative model of cognition and have been applied to number of applications, this approach is not very easy to apply because of the huge number of parameters to tune. The ten given principles are very general and no associated tool exists. No guide is given to indicate if the use of swarming systems is more relevant than conventional cognitive techniques for designing the current application or problem.

De Wolf [15] has defined a full life-cycle methodology based on the Unified Process customized to explicitly focus on engineering macroscopic behaviour of such kind of systems. This customization takes place in the following steps of the process:

- After the requirements analysis done, one checks if an autonomous behaviour is needed, if the available information is distributed, if the system is subject to high dynamics such as failures and frequent changes;
- In the design phase, general guidelines or principles, reference architectures, decentralised mechanisms allowing coordination between agents to achieved desirable macroscopic properties, have to be used to design self-organising emergent MAS. In that sense, De Wolf proposes an initial catalogue including the most widely used coordination mechanisms such as digital pheromones, gradient fields, market based coordination, and tag based coordination. Furthermore, he proposes "Information flow" as a design abstraction which enables designing a solution independently of the coordination mechanism.
- In the verification and testing phase, he combines agent-based simulations with scientific numerical algorithms for dynamical systems design. More detailed are given in the challenge 3 of this paper.

3.2 Engineering Adaptive Multi-agent Systems: ADELFE

ADELFE [5] is a methodology devoted to software engineering of adaptive multi-agent according to the AMAS approach. ADELFE enables the development of software with emergent functionality and consists of a notation based on UML (*Unified Modelling Language*) and AUML (*Agent-UML*) [34], a design process

[5] ADELFE is a French acronym for "Atelier de Développement de Logiciels à Fonctionnalité Émergente".

based on the RUP (*Rational Unified Process*), a platform made up of a graphical design tool called OpenTool and a library of components that can be used to make the application development easier.

The design process (see figure 2) covers all the phases of a classical software design (from the requirements to the deployment) in adding some specific steps to design adaptive systems. OMG's SPEM (*Software Process Engineering Metamodel*) has been used to express the ADELFE process and the SPEM vocabulary is used to expound the methodology: WorkDefinitions (WDi), Activities (Aj) and Steps (Sk).

ADELFE : Design Methodology

- Final requirements. The environment of the system is central in the AMAS theory; this is due to the fact that the adaptation process depends on the interactions between the system and its environment. This characteristic has led to the addition of one Activity (A6) and one Step (A7-S2) in the "Final Requirements" WD2. Designers must characterize the environment of the system by qualifying it as being accessible or not, deterministic or not, dynamic or static and discrete or continuous. These terms represent a help to later determine if the AMAS technology is required or not to build the studied system (A11). At this point, designers must also begin to think about the situations that can be "unexpected" or "harmful" for the system because these situations can lead to NCS at the agent level. Therefore, the determination of use cases has been modified to take this aspect into account (S2).
- Analysis. The use of AMAS theory is not a solution fitted to every application. For that reason, ADELFE provides an interactive tool (A11) to help a designer to decide if the use of the AMAS theory is required to implement his application. ADELFE does not assume that all the entities defined during the final requirements are agents. Therefore, this methodology focuses on the agents identification (A12) and some guidelines are then provided to help designers to identify agents [35]. A Step (S3) has also been added concerning the study of agents relationships.
- Design. Agents being identified and their relationships being studied, designers have now to study the way in which the agents are going to interact (A15) thanks to protocol diagrams. ADELFE also provides a model for designing cooperative agents (A16), following the agent architecture presented in section 2.5. The global function of a self-organising system is not coded; designers have only to code the local behaviour of the parts composing it. ADELFE provides some generic cooperation failures such as incomprehension, ambiguity, uselessness or conflict. Designers must fill up one table per NCS to give the name of each NCS, its generic type, the state in which the agent must be to detect it, the conditions of its detection and what actions the agent must perform to deal with it. A new Activity (A17) of fast prototyping based on finite state machine has been added to the process. It enables designers to verify the behaviour of the built agents. Now simulations tools is included into ADELFE to complete the life cycle of its development process [36,37].

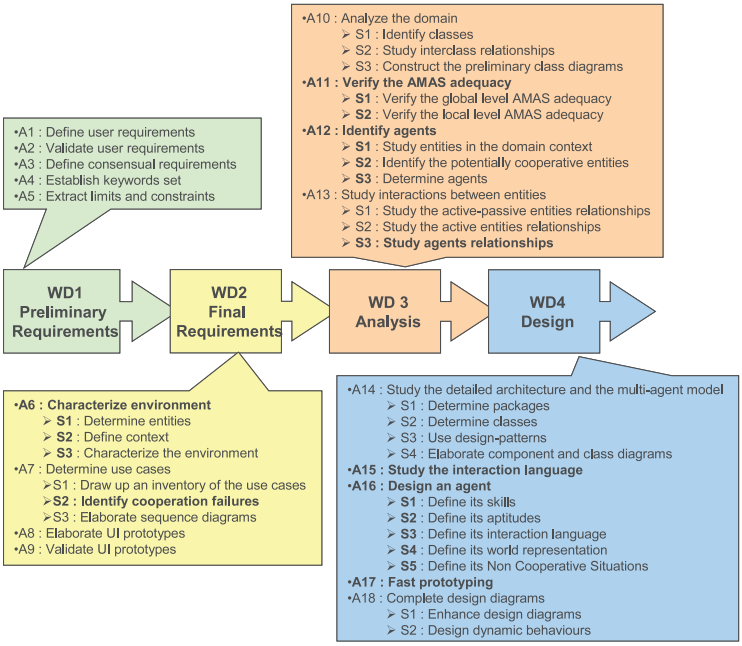

Fig. 2. ADELFE process

Tools Linked with ADELFE. Even if ADELFE is suited to develop applications based on the AMAS technology, it does not assume that the designer is specialized in this field. Therefore, some additional notations are provided as well as some tools to help or guide the designer throughout the process application [38]:

1. A tool enabling to know if the use of the AMAS technology is useful to implement the target system. Eleven questions are asked to designers using a graphical interface. This adequacy is studied at two levels: the global one (system) with 8 questions and the local one (components) with 3 questions. A designer uses a slider to answer a question and to give a rate among twenty possibilities ranging from "yes" to "no". His answers are then analysed by the support decision tool. The answers of ADELFE regarding the global level and the local one are then given in a graphical tool and an interpretation of the results can also be obtained.
2. An interactive tool which describes the process and helps the designer to apply it (it can be downloaded at http://www.irit.fr/ADELFE). The first functionality of the ADELFE interactive tool is to be a guide by describing the process; each activity or step of the process is depicted and exemplified by applying it to a timetabling problem (ETTO) [35]. The interactive tool also provides a means to support the adopted notations and draw the needed diagrams by integrating OpenTool which has been modified for ADELFE. It checks the project consistency by displaying what stages (activities or steps)

can be done depending on what has been already done or what documents have been produced yet.

3. OpenTool, a graphical modelling tool supporting the UML notation and embedded in the ADELFE toolkit. It enables applications modelling while assuring that the produced models are valid. As some lacks exist in the UML notation to deal with the specific modules composing a cooperative agent, nine stereotypes have been defined to show how an agent is formed and/or how its behaviour is expressed («cooperative agent» «characteristic» «skill», «aptitude», «representation», «interaction», «perception», «actions» and «cooperation»). On the other hand, to model interaction protocols between agents AUML interaction protocol model has been extended and included in OpenTool functionalities. OpenTool has also been modified to enabling expression of cooperation failures. In the fast prototyping stage (A-17), agents' behaviours are simulated using a functionality of OpenTool which requires a dynamic model (state-chart) for each simulated entity (object or agent). As agents' behaviours are modelled as AIP protocol diagrams and a method was proposed to transform a protocol diagram (a particular generic sequence diagram) into a state-chart that OpenTool is able to simulate.

4 Challenge 3 : A Validation Framework for the Designer of Complex Systems

It is quite obvious that the software validation phase, requested by industrials and end-users, is necessary before its commercialization. So, validation of self-organising applications is, even more, a mandatory step during development. In software engineering, there are often many validation activities but in this paper we focus on the global behaviour validation of the system which consists in verifying that the system complies to the desired function. Validation is not a new axis in computer science, but self-organising systems lead to new challenges not yet taken into account by classical methods. In large scale dynamic and adaptive systems such as self-organising systems, the methods, techniques and tools for validation are still in a research phase [39]. In general, formal methods [40] for validation, such as model checking, theorem proving... are adequate for checking/proving desired properties of the system when the code is showing the following properties: it is static and it runs in well-known environments. A static code is a code which does not evolve and there is no learning at this level. A well-known environment means that the system does not face unexpected events or unexpected scenarii.

4.1 Related Works in Multi-agent Systems

The question of validation becomes more and more crucial in self-organising MAS and some works attempts to deal with it. Tom De Wolf et all [41] use simulation-based scientific analysis for designing self-organising systems achieving the required system behaviour. They combine realistic agent-based simulation and existing scientific numerical analysis algorithms to design a system

simulation. The main phase in the design process is to identify macroscopic properties desired at the system level, macroscopic variables for measuring the macroscopic properties and define microscopic variables for each macroscopic one. Then when an analysis algorithm is chosen, simulations are launched so as to analyse the global behaviour of the system in terms of desired properties linked to specific parameters. Parameters are adjusted iteratively until the systems exhibits a satisfactory behaviour. The difficulty in this approach is to define the different variables and express the link between the macro and micro levels.

Bruce Edmonds has shown in several papers [42,43] that formal methods are insufficient to show the reliability of self-organised systems. He proposes to use an experimental method to produce reliable self-organising systems and mixes in his approach engineering and adaptation. After the process of construction which constructs the multi-agent systems from the agents, the design process consists in adaptation cycles. A cycle begins with a test and comparison of the global behaviour of the system and the desired global behaviour. If it is not satisfying the system adapts to change its global behaviour. This cycle stops when the produced global behaviour fits the desired one. The validation is done by experiments and is considered by the authors as the sole mean at this time.

4.2 Validation of AMAS

In ADELFE, the reliability of the global behaviour of the system, ensured by the AMAS theory, is verified essentially at the design phase. In self-organising system, the desired function cannot always be well defined, for example: what is the global function of the Internet? What is the global function of a crowd? By consequence the automatic verification is not always possible and must be done by the designer. In ADELFE, this functional adequacy is checked at the end of the design but also during the design. Our aim is to give more tools to automate the verification-update cycle. We are very close to Edmonds's approach. The tests realized by simulation help to enhance the system and to improve the functional adequacy, i.e. to verify that the system fits the desired function.

As explained in [11], we can consider in agent-based software engineering that the object conceptual level and the agent conceptual level in the system design process overlap. The test phase of the code realized with the targeted programming language is done in parallel with the agent design phase (see figure 3). So, in ADELFE, what we call *Living Design* is defined by the link between design and test phases of the two processes. Namely, *Living Design* means "construct agents during run-time". Therefore, the designer is like a biologist who studies the behaviour of living creatures and who can modify its model according to his observations. For doing this, simulation is used in order to help the designer to develop the agents of the system by observing the system at the global level.

As we have say before, applying the AMAS theory consists in enumerating, according to the current problem to solve, all the NCS that can appear during the system functioning and then defining the actions the system must apply to return to a cooperative state. Currently, during the preliminary requirements phase, ADELFE provides tools to express NCS in the use case diagrams. During

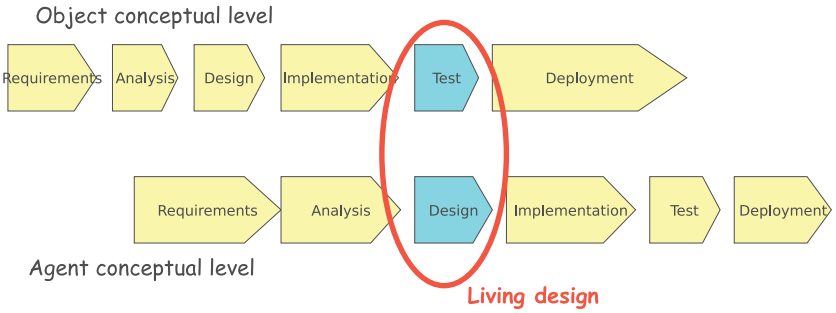

Fig. 3. Living Design

the design phase, it is possible to find if some deadlocks can take place within an interaction protocol, or if some protocols are useless or inconsistent. The protocol diagram notation has been extended to express these situations. Thus, the behaviour of several agents could be judged in accordance (or not) with the sequence diagrams described in the analysis phase. However, the core difficulty lies in identifying NCS and in helping the designer to find all these NCS. Simulation is used in ADELFE to help designers to find the correct behaviours of the agents during the design stage: by simulating a simplified system and observing it during execution, the behaviour of agents can be modified and improved. Currently, our work focuses only on situated multi-agent systems and not yet on communicative ones. The main reason for such a choice was that the observation of the behaviour in an environment is easier to be judged by an observer. Simulation enables to automatically identify these situations during execution of a prototype of a targeted MAS. A model of cooperative agents [37] is implemented under the *SeSAm* platform by using an architecture inspired from the subsumption architecture proposed by Brooks and reusing the notion of priority between the *(conditions, actions)* couple. This behaviour can be expressed with a set of behavioural rules which follow this pattern:

if **premise** then **consequent**

where **premise** is a logical predicate made up of elements coming from agent perceptions or characteristics, and **consequent** activates one of the possible actions this agent may perform. The difference between Brooks' architecture and ours is that our agents have representations. The cooperative agent model [37] automatically detects some NCS during the execution of a simulated MAS and shows where and when NCS appear. The designer has then to modify the agent behaviour.

In our more recent work [36], this goes a step further as during a simulation cycle, an agent has the ability to *self-design* its behaviour considering that *(i)* all the rules needed to design the decision process are given by a designer (that is the agent does not learn new rules during the process), *(ii)* the set of given rules is complete and correctly written and *(iii)* the system interacts with a

dynamic environment. The Self-Design Behaviour Module (SDBM) inside an agent is implemented as an adaptive MAS. Behavioural rules forming it have to collectively adapt to the agent's environment and are then considered as agents. They have to self-organise in order to find the best hierarchy of rules that is to say the most efficient behaviour for the agent it belongs to. This work is on going but we obtained first encouraging results [36].

Because the systems are adaptive, it is necessary to validate them in a dynamic environment. The number of states and of events perceived by the system must be very important and the validation can not currently be formal. The formal validation can be done on the agent code but not yet at the global level, so currently on these systems only partial formal validation can be done and the global behaviour can be verified only by simulation. The role of the designer/observer is fundamental because he participates to the co-construction of the system. He plays the role of an environment which interacts with the system and causes the change of the system behaviour.

5 Conclusion

We have presented the three main challenges for engineering systems with emergent functionalities will be confronted with, as well as current investigated leads and work relating to the use of self-organisation and emergent phenomena.

1. System control related problems can be partially solved by providing the system with capability to self-adapt to the environment. Common decentralized mechanisms used to achieve such a control are inspired by existing natural systems (ants colony, collective movements) or social-related behaviours (cooperation, competition). This is the case for the AMAS theory which is based on an environmental constraints driven process and enables engineering systems whose parts self-organise according to local cooperative criteria.
2. In order to support those new ways to design complex systems, new tools and methodologies have to focus on local behaviours, environment characterization and emergent phenomena. Unfortunately, the few existing methodologies are yet in the research domain and/or incomplete (no deployment and maintenance phases). Moreover, system design analysis is still strongly focused on global ends analysis which cannot fit with some application requirements: for instance, in *Ambiant Intelligence* it is not possible to fully specify what the system has to do.
3. Given that emergence and self-organisation had not seriously been studied as hard science subject, classical formal methods are not suitable. Engineered complex systems verification and validation can only be achieved using simulation-based approaches. Nowadays, the most reliable way consists in iteratively improving the designed system using mathematical tools (statistical analysis, behavioural parameters optimisation) or semi-autonomous adaptive programming (*Living Design*).

Emergence and self-organisation have only recently been considered as serious alternatives in industrial software engineering. As anyone can notice by reading

strategic agendas of some European platforms (ARTEMIS, eMobility, EPoSS), the main displayed concerns about these "new paradigms" are validation and verification aspects:

> "*These system-design principles seem to be compatible with the good average-case performance. However, these often conflict with a design's predictability.*" (Artist2 Network of Excellence, 2006[6])

Nevertheless, in our opinion, industry does not really have a choice: as software becomes more complex, this approach is the only viable option currently known. True artificial complex systems will thus be built using emergence and self-organisation: Ambient Intelligence, Swarm Robotics, Autonomous Computing, e-Health-care, Computational Biology...

Another major effort has to be done towards methodologies supporting (enabling) pure local analysis without any need to specify what the system has to do or underspecified system. As a matter of fact, the core of the complex system engineering problem remains the lack of accepted theories (even non complete ones) of emergence and self-organisation. Without such a fundamental key, it will be difficult to legitimate and disseminate this approach, as well as to promote and explain any future successful "killer applications".

References

1. Dréo, J., Pétrowski, A., Siarry, P., Taillard, E.: Metaheuristics for Hard Optimization. Springer, Heidelberg (2006)
2. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge (1998)
3. Lawrence, J.: Introduction to Neural Networks. California Scientific, Nevada City (1993)
4. Di Marzo Serugendo, G., Karageorgos, A., Rana, O.F., Zambonelli, F. (eds.): Engineering Self-Organising Systems, Nature-Inspired Approaches to Software Engineering. In: Di Marzo Serugendo, G., Karageorgos, A., Rana, O.F., Zambonelli, F. (eds.) ESOA 2003. LNCS (LNAI), vol. 2977, Springer, Heidelberg (2004)
5. Brueckner, S., Di Marzo Serugendo, G., Karageorgos, A., Nagpal, R. (eds.): Engineering Self-Organising Systems, Methodologies and Applications. In: Brueckner, S.A., Di Marzo Serugendo, G., Karageorgos, A., Nagpal, R. (eds.) ESOA 2005. LNCS (LNAI), vol. 3464, Springer, Heidelberg (2005)
6. Brueckner, S., Di Marzo Serugendo, G., Hales, D., Zambonelli, F. (eds.): Engineering Self-Organising Systems. In: Brueckner, S.A., Di Marzo Serugendo, G., Hales, D., Zambonelli, F. (eds.) ESOA 2005. LNCS (LNAI), vol. 3910, Springer, Heidelberg (2006)
7. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, Oxford (1999)
8. Di Marzo Serugendo, G., Martin-Flair, J.-P., Jelasity, M., Zambonelli, F. (eds.): Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2007, Boston, MA, USA, July 9-11, 2007. IEEE Computer Society Press, Los Alamitos (2007)

[6] http://www.artist-embedded.org/artist/-Research-.html

9. Di Marzo Serugendo, G., Gleizes, M.P., Karageorgos, A.: Self-Organization and Emergence in Muli-Agent Systems. The Knowledge Engineering Review 20(2), 165–189 (2005)
10. Wolf, T.D., Holvoet, T.: Emergence versus self-organisation: Different concepts but promising when combined. In: Brueckner, S.A., Di Marzo Serugendo, G., Karageorgos, A., Nagpal, R. (eds.) ESOA 2005. LNCS (LNAI), vol. 3464, pp. 1–15. Springer, Heidelberg (2005)
11. Capera, D., Georgé, J.P., Gleizes, M.P., Glize, P.: The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents. In: 1st International TAPOCS Workshop at IEEE 12th WETICE, pp. 383–388. IEEE, Los Alamitos (2003)
12. Georgé, J.P., Gleizes, M.P.: Experiments in Emergent Programming using Self-organizing Multi-Agent Systems. In: Pěchouček, M., Petta, P., Varga, L.Z. (eds.) CEEMAS 2005. LNCS (LNAI), vol. 3690, pp. 450–459. Springer, Heidelberg (2005)
13. Zambonelli, F., 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)
14. Parunak, H., Brueckner, S.: Engineering Swarming Systems. Kluwer, Dordrecht (2004)
15. De Wolf, T.: Analysing and engineering self-organising emergent applications. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium (2007)
16. Goldberg, D.: Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley, Reading (1989)
17. Holland, J.: Adaptation in Natural and Artificial Systems. MIT Press, Cambridge (1992)
18. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysic 5, 115–133 (1943)
19. Rosenblatt, F.: The Perceptron: probabilistic model for information storage and organization in the brain. Psychological Review 65, 386–408 (1958)
20. Kohonen, T.: Self-Organising Maps, vol. 30. Springer, Heidelberg (2001)
21. Weiß, G.: Multiagent Systems, A modern Approach to Distributed Artificial Systems. MIT Press, Cambridge (1999)
22. Wooldridge, M.: An introduction to multi-agent systems. John Wiley & Sons, Chichester (2002)
23. Mano, J., Bourjot, C., Lopardo, G., Glize, P.: Bio-inspired Mechanisms for Artificial Self-organised Systems. Informatica 30(1), 55–62 (2006)
24. Hassas, S., Castelfranchi, C., Di Marzo Serugendo, G., Karageorgos A.: Self-Organising Mechanisms from Social and Business/Economics Approaches. Informatica 30(1) (2006)
25. Di Marzo Serugendo, G., Gleizes, M.P., Karageorgos, A. (eds.): Self-organisation in MAS - Tutorial at the 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2005), Budapest, Hungary, September 15 (2005)
26. 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)
27. Ferber, J.: Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. Addison-Wesley, Reading (1999)

28. 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)
29. Axelrod, R.: The Evolution of Cooperation. Basic Books (1984)
30. Huberman, B.: The performance of cooperative processes. MIT Press / North-Holland (1991)
31. Heylighen, F.: Evolution, Selfishness and Cooperation; Selfish Memes and the Evolution of Cooperation. Journal of Ideas 2(4), 70–84 (1992)
32. 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)
33. Henderson-Sellers, B., Giorgini, P.: Agent-Oriented Methodologies. Idea Group Pub., NY (2005)
34. Odell, J., Parunak, H., Bauer, B.: Representing Agent Interaction Protocols in UML. Springer, Heidelberg (2001)
35. Bernon, C., Gleizes, M.P., Peyruqueou, S., Picard, G.: Adelfe: a methodology for adaptive multi-agent systems engineering. In: Petta, P., Tolksdorf, R., Zambonelli, F. (eds.) ESAW 2002. LNCS (LNAI), vol. 2577, pp. 156–169. Springer, Heidelberg (2003)
36. Lemouzy, S., Bernon, C., Gleizes, M.P.: Living design: Simulation for self-designing agents. In: Multi-Agent Systems and Simulation (MAS&S) Workshop at ESM 2007 (to be published)
37. Bernon, C., Gleizes, M.P., Picard, G.: Enhancing self-organising emergent systems design with simulation. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS (LNAI), vol. 4457, pp. 284–299. Springer, Heidelberg (2007)
38. Bernon, C., Camps, V., Gleizes, M.P., Picard, G.: Tools for self-organizing applications engineering. In: Di Marzo Serugendo, G., Karageorgos, A., Rana, O.F., Zambonelli, F. (eds.) ESOA 2003. LNCS (LNAI), vol. 2977, pp. 283–298. Springer, Heidelberg (2004)
39. Slaby, J., Welch, L., Work, P.: Toward certification of adaptive distributed systems. In: Real-time and Embedded Systems Workshop (2006)
40. Clarke, E.M., Wing, J.M.: Formal methods: state of the art and future directions. ACM Comput. Surv. 28(4), 626–643 (1996)
41. Wolf, T.D., Holvoet, T., Samaey, G.: Development of self-organising emergent applications with simulation-based numerical analysis. In: Engineering Self-Organising Systems, pp. 138–152 (2005)
42. Edmonds, B., Bryson, J.: The insufficiency of formal design methods - the necessity of an experimental approach - for the understanding and control of complex mas. In: AAMAS, pp. 938–945 (2004)
43. Edmonds, B.: Engineering self-organising systems, methodologies and applications. In: Brueckner, S.A., Di Marzo Serugendo, G., Karageorgos, A., Nagpal, R. (eds.) ESOA 2005. LNCS (LNAI), vol. 3464, Springer, Heidelberg (2005)