Emergence and Software development Based on a Survey of Emergence Definitions

Joris Deguet[1 2], Laurent Magnin[2], and Yves Demazeau[1]

[1] Laboratoire Leibniz, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France
joris.deguet@imag.fr, yves.demazeau@imag.fr,
[2] DIRO, Université de Montréal, Montréal, (Québec) H3C 3J7, Canada
magnin@iro.umontreal.ca

1 Introduction

Emergence, a concept that first appeared in philosophy [18, 17], has been widely explored in the domains of Multi-agent Systems (MAS) and Complex Systems [25, 13, 14, 5, 4, 6, 2, 12] and is sometimes considered to be the "key ingredient that makes complex systems complex" [24].

On January 30th 2006, we made a one-keyword query for "emergence" papers on computer science specific engines and generalist scientific engines. We retrieved impressive amounts of documents:

Table 1

Search Engine	Number of results
ACM	1606
IEEE	783
CiteSeer	8596
ScholarGoogle	675000

However, there is still a lack of well defined Emergence Based Engineering (EBE) methodologies. Before building such methodologies, we have to look at what implies emergence into software development. Since there is multiple definitions of emergence, we will build our study based on five papers[3] that match the following criteria:

- Emergence definition is the primary goal
- It contains a significantly different (and possibly contradictory) approach from other selected papers

[3] A previous paper [8] presents a deeper analysis of those emergence definitions.

2 Emergence definitions, usual software development and Emergence Based Engineering

Emergence Based Engineering can be defined as "efficient methodologies to build systems that will produce emergent (and useful) phenomena". Conceiving Emergence Based Engineering approaches might not be an easy task, so before going further, let see if we could apply the usual software development methodologies to achieve that goal.

In general, how to design a system that will produce phenomena? In fact, as software analysts and developers, we do not want to produce random phenomena (*i.e.* random system behaviours) in general, but a specific set of well defined phenomena specified through requirements. To obtain such phenomena, we know, thanks to our understanding of how a computer handle a code, that we have to design and code the system in a specific way. In other words, we (need to) understand the causality between the code and the results of its execution.

So, is it possible to use traditional design and coding approaches for Emergence Based Engineering? We will in the following sections study that question through definitions of emergence.

2.1 Detection and emergence

A first definition of emergence is provided by Bonabeau and Dessalles [4]. Given the two following notions:

detector	defined as "any device which gives a binary response to its input"
relative complexity	$C(S\|D,T)$ of a system S "where D is a set of detectors and T a set of available tools that allow to compute a description of structures detected through D" which corresponds to the difficulty to describe the system given T and D.

Emergence happens when between time t and $t + \Delta t$, two events happen:

1. a detector D_k becomes activated
2. $C_{t+\Delta t}(S|T, D_1, \ldots, D_{k-1}, D_k) < C_t(S|T, D_1, \ldots, D_{k-1})$

This property is likely to happen in a hierarchy of detectors when an upper level entity summarizes states of a lower level. Thus emergence is the apparition of a synthetic entity.

One widely shared feature of emergence definitions is the existence of levels. Bonabeau and Dessalles do not assume levels *a priori* in the definition but suggest that this is a condition *sine qua non* for the complexity discontinuity to happen.

No assumption is made about the system under detection, therefore one can apply this criterion on both artificial and natural systems as long as detection is possible.

Emergence Based Engineering's Implications

Usual software development assumes that the expected phenomena produced by the software under development can be predicted directly from the code; in order to design and write that code. Here, we cannot predict the nature of the phenomena statically, but by detectors that can be used only after coding of the system, at runtime... Therefore, usual software development is not suited to design systems that produce emergent phenomena as defined by Bonabeau and Dessalles.

2.2 The emergence test

The first definition focused on an observer modelled by a detection apparatus made emergence somehow "subjective" as the complexity measure depends on this apparatus. However, once the observer is defined, emergence only depends on the perceived behaviour. The emergence test [22, 21] introduces the consideration of the system's design in addition to its behaviour.

This *emergence* test involves a system designer and an observer (possibly the same person). Then if the following three conditions hold, the emergence tag is conferred:

Design	The system has been constructed by the designer by describing *local* elementary interactions between components in a language L_1
Observation	The observer is *fully aware* of the design, but describes *global* behaviour and properties over a period of time, using a language L_2
Surprise	The language of design L_1 and the language of observation L_2 are distinct, and the causal link between the elementary interactions programmed in L_1 and the behaviours observed in L_2 is *non-obvious* to the observer, who therefore experiences surprise.

We can consider Bonabeau and Dessalles' D and T as words and syntax of an observation language L_2.

The introduction of the design language L_1 has two important consequences:

1. Emergence happens between the design and the observation. This defines a *design-to-behaviour emergence.*
2. Existence of L_1 restricts the application of this criterion to artificial systems.

Emergence happens when observation and design appear loosely coupled to the observer.

In the field of decentralized artificial intelligence, Demazeau and Müller [11] have made a similar distinction between *internal* and *external* descriptions of agents where internal description refers to the real architecture of an agent and external description refers to its externally perceived behaviour.

Emergence Based Engineering's Implications

Classical software engineering requires that the designer of a system coded in language L_1 can predict the future behaviour of its system, described by language L_2 (used to express the requirements of the system). This is in contradiction with the emergence test as L_2 is not likely to be both predictable and surprising. Also, since "emergence happens between the design and the observation", it is *de facto* not possible to conceive an emergent phenomenon during the design phase... Then again, classical software development is not suited to design emergent behaviours.

2.3 Simulation emergence

Making the parallel between intelligence and emergence as subjective notions defined by tests can lead to controversy. One answer could be to consider that emergence happens when a large number of scientists agree that it does. Another answer is to make the definition objective. Simulation emergence is such an attempt, focused on the simulation domain.

In Darley [7] we find this definition:

> "A true emergent phenomenon is one for which the optimal means of prediction is simulation."

The author defines two means of prediction depending on n the size of a system:

- $s(n)$: the optimal "amount of computation required to simulate a system, and arrive at a prediction of the given phenomenon".
- $u(n)$: stands for "deeper level of understanding", the way we try to avoid computation by "a creative analysis", $u(n)$ is the amount of computation required by this method.

Then the system will be considered as emergent iff $u(n) \geq s(n)$ i.e. direct simulation is optimal relative to the "amount of computation" measure.

The key issue is to understand what a simulation is. Among all the ways to derive the phenomenon in a computable manner, some are simulations, others are "shortcuts". Then optimality of simulation is equivalent to the absence of "shortcuts".

An interesting point is that both authors address the question of emergence's decidability:

- In Bedau's formulation: "One might worry that the concept of weak emergence is fairly useless since we generally have no proof that a given macro-state of a given system is underivable without simulation."
- With Darley's words "Can we determine, for a given system, whether or not it is emergent ?".

If we reformulate as "the global behavior is optimally obtained by running a system made of interacting micro agents", it provides a natural way to apply the definition to multi-agent based simulations.

Emergence Based Engineering's Implications

As already said in the "Detection and emergence" section, usual software development assumes that the expected phenomena produced by the software under development can be predicted directly from the code. Here again, since "a true emergent phenomenon is one for which the optimal means of prediction is simulation", the prediction of the behaviour is not possible directly from the code (or at a too high cost), which is in contradiction with usual software development.

2.4 Downward causation and emergence

Bedau has defined *weak* emergence with respect to the *strong* emergence based on *downward causation*. This view is illustrated by Timothy O'Connor [20]:

> "to capture a very strong sense in which an emergent's causal influence is irreducible to that of the micro-properties on which it supervenes; it bears its influence in a direct *downward* fashion, in contrast to the operation of a simple structural macro-property, whose causal influence occurs via the activity of the micro-properties which constitutes it."

In order to achieve *downward causation*, Sawyer [23] proposes that:

1. "as in blackboard systems, the emergent frame must be represented as a data structure external to all of the participating agents"
2. "all emergent collective structures must be internalized by each agent, resulting in an agent-internal version of the emergent."
3. "This internalization process is not deterministic and can result in each agent having a slightly different representation."

We believe that L_1 and L_2 are of significant interest to clarify this issue. It sounds natural to us to consider that everything with causal powers in an artificial system lies in the L_1 design language as it must live within algorithm. Thus even if a data structure exists out of the agents at a macro level, it belongs to the design language. Then L_2 to L_1 causal power is impossible.

Until here we might have mixed design/observation with micro/macro as it is often the same: We conceive agents and we are very happy to show their collective behavior to colleagues. However, it can be interesting to distinguish the micro/macro from design/observation.

Sawyer's definition is based on the existence of a macro entity external to micro agents. This existence might provide causal powers to this entity on agents. Therefore it allows a *macro to micro causation* we can consider as *downward* as scale decreases. However, this is different from O'Connor's view as agents do not constitute the macro entity.

Emergence Based Engineering's Implications

"Downward causation" applied to code and behaviour means that the code/algorithm is determined by the behaviour, not the programmer/designer. In others words, we give to the "machine" a description of the expected behaviour and we get some code... That is not usual software design process (again), but kind of machine learning.

2.5 Grammar emergence

This last definition of emergence is specific as its scope is limited to systems expressed in a particular grammar model. This model provides intuitive definitions for micro/macro and design/observation distinctions.

Kubik [16] has proposed an approach based on "the whole is more than the sum of its parts" as inspiration and grammars as a modelling tool.

The key idea is to define a "whole" language and a "sum of the parts" language. From an initial configuration, a language is obtained by rewriting using production rules. For a given set of rules P_i, the corresponding language is noted $L(P_i)$.

We can sum up the proposal as follows:

$$\underbrace{L(\bigcup_i P_i)}_{Whole} \underbrace{\supset}_{More} \underbrace{superimposition_i}_{Sum} \underbrace{(L(P_i))}_{Parts}$$

We do not give the definition of the *superimposition* operator here.

Emergence is the case of a configuration being in the whole language but not in the sum of parts. The first is obtained by putting all parts together and deriving configurations, the last by deriving configuration for every part separately and putting results together afterward. Putting together is the way we get a macro entity from micro ones, and derivation is the way to get the language (L_2) we observe from the rules (L_1) we designed.

When someone hears "the whole is more than the sum of its parts", he or she might reply very fast that a system *is* composed of its parts and therefore cannot be more. To go beyond this triviality, Kubik's elegant idea is to switch micro/macro with design/observation. This makes things comparable as Kubik defines his gap between two set of configurations (similar to L_2 and a L_2'), at the observation level.

Kubik's idea is close to an informal definition of emergence from [10] stated in the VOWELS framework [9] for multi-agent systems (MAS). This framework suggests a description of such systems as agents (A) in their environment (E), using interactions (I) forming an organization (O). Then the pseudo equation from [10]:

$$MAS = A + E + I + O + Emergence$$

can be seen as:

$$\underbrace{L(MAS)}_{Whole}\underbrace{\supset}_{More}\underbrace{\sum_{v\in vowels}}_{Sum}\underbrace{(L(v))}_{Parts}$$

with VOWELS as an alternate micro partition of a macro MAS.

Emergence Based Engineering's Implications

"The whole is more than the sum of its parts." Usually the design of a complex system is based on its decomposition by the designer to decrease the relative complexity of its subparts. However, it seems that based on that emergence definition when we decompose the system we add a new global complexity that is bigger than what we get by designing the subparts.

Also, usual software development is strongly based on incremental testing which consist in testing first small parts of code (unit testing), then larger parts, then the complete integration of modules. Here, the general behaviour of the system will radically be different than the behaviour of its parts, invalidating their individual validations. So, again, usual software development, which is heavily based on decomposition, is not well suited for the kind of "emergent" system we would like to produce.

3 What are the Emergence Based Engineering alternatives?

Based on what we saw in the last chapter, it is not possible to achieve "classical" emergent programming. To summarize that chapter, when a software developer design and code a system that produces phenomena, if she understands how such phenomena will be produced they cannot be qualified as emergent. At the opposite if she does not know what she achieves, she also does not know what behaviour her program will exhibit... In other words, lack of causality understanding, which is recurrent in emergence definitions, is in opposition with usual programming methodologies.

So, to design a system that will produce a given but also emergent phenomenon, we have to employ different methodologies than usual software development. The main idea is to implement or generate the system without knowing "how it works". We can achieve that goal by at least three approaches:

- By imitating phenomena usually considered as emergent. For example, by implementing the mechanisms of an "ants foraging like algorithm" we could expect the same global behaviour for our system, without having to understand why that behaviour appears;

- By using an incremental design process. First step, we implement a generic system that will produce a behaviour. Based on such behaviour, we try to adapt the system to make it producing a behaviour that will be closer to the behaviour we ultimately expect. Then, we analyse again the produced behaviour, try to adapt the system, etc. ;
- By developing self-adaptive systems. In that case, we could understand how the (meta-)system will be able to modify itself to generate new behaviours when the context changes, but we cannot know in advance what solutions it will produce.

4 Conclusion and perspectives

We have seen that Emergence Based Engineering needs new software development approaches. To do so, we suggest to use 1) insights provided by definitions and mechanisms suggested by widely accepted emergence examples (social animals, markets), or 2) machine learning techniques (off-line or embedded).

However, still then remains a lot of issues. How to apply those approaches so to generate in fact emergent phenomena? What will be the differences between "non emergent" machine learning and "emergent" machine learning?

But before going further, we need an unified and computable definition of "computer science emergence" to validate the "emergenceness" of such methodologies. In [8], we have isolated a minimal setting, small as definitions are significantly different.

By going through the definitions [8], we have noticed that emphasis is usually put on the criterion proposed. However, for a computational definition, we think the following points should be refined:

- How do we apply levels on existing systems?
- Can we tag a phenomenon as emergent in a computable way?

We might also explore to what extent a specific definition of emergence is linked with definitions of self-organization or complexity and other terms we usually meet in the field of complex agent networks.

Nonetheless, the reason we wanted a Emergence Based Engineering is the "much from little" idea that Holland has associated to emergence [15]. Indeed, since software systems, in particular multi-agent systems, are going bigger and more complex, reducing the size or the complexity of what is needed to build those systems will become more and more essential. Therefore, Emergence Based Engineering (EBE) sounds like an appealing research track.

5 Aknowledgements

We thank the French Government's Direction Générale de l'Armement (DGA) for supporting the research reported in this paper.

References

1. P. Angeline. *Advances in Genetic Programming*, chapter Genetic Programming and Emergent Intelligence. MIT Press, 1994.
2. A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 1999.
3. C. Bernon, M. Gleizes, S. Peyruqueou, and G. Picard. Adelfe, a methodology for adaptive multi-agents systems engineering. In *ESAW 2002*, 2002.
4. E. Bonabeau and J. Dessalles. Detection and emergence. *Intellectica*, 2(25), 1997.
5. E. Bonabeau, J. Dessalles, and A. Grumbach. Characterizing emergent phenomena. *Revue Internationale de Systémique*, 1995.
6. E. Bonabeau, G. Theraulaz, J. Deneubourg, N. Franks, O. Rafelsberger, J. Joly, and S. Blanco. A model for the emergence of pillars, walls and royal chambers in termite nests. *Philosophical Transactions: Biological Sciences*, 1998.
7. V. Darley. Emergent phenomena and complexity. In R. Brooks and P. Maes, editors, *Artificial Life 4*, pages 411–416, 1994.
8. J. Deguet, Y. Demazeau, and L. Magnin. Elements about the emergence issue, a survey of emergence definitions. *ComPlexUs, International Journal on Modelling in Systems Biology, Social, Cognitive and Information Sciences*, 2006.
9. Y. Demazeau. Steps towards multi-agent oriented programming. In *1st International Workshop on Multi Agent Systems*, 1997.
10. Y. Demazeau. Voyelles. Technical report, CNRS, 2001.
11. Y. Demazeau and J. Muller. From reactive to intentional agents. In *Decentralized Artificial Intelligence 2*. Elsevier, 1991.
12. J. Deneubourg, A. Lioni, and C. Detrain. Dynamics of aggregation and emergence of cooperation. *Biol. Bulletin*, 2002.
13. J. Deneubourg, G. Theraulaz, and R. Beckers. Swarm-made architectures. In *Toward a practice of autonomous systems*, 1992.
14. N. Gilbert. Emergence in social simulation. In *Artificial societies: The computer simulation of social life*. UCL Press, 1995.
15. J. Holland. *Emergence: From Chaos to Order*. Perseus Books, 1997.
16. A. Kubik. Toward a formalization of emergence. *Artificial Life*, 9, 2003.
17. G. Lewes. *Problems of Life and Mind*. Trubner and Company, 1874.
18. J. Mill. *System of Logic*. John W. Parker, 1843.
19. J. Muller. Emergence of collective behaviour and problem solving. In *ESAW*, 2003.
20. T. O'Connor. Emergent properties. *American Philosophical Quaterly*, 1994.
21. E. Ronald and M. Sipper. Surprise versus unsurprise: Implications of emergence in robotics. *Robotics and Autonomous Systems*, 2001.
22. E. Ronald, M. Sipper, and M. Capcarrère. Design, observation, surprise! a test of emergence. In *Artificial Life 5*, pages 225–239, 1999.
23. R. Sawyer. Simulating emergence and downward causation in small groups. In *MABS*, 2001.
24. R. Standish. On complexity and emergence. *Complexity International*, 2001.
25. L. Steels. Towards a theory of emergent functionality. In *From Animals to Animats SAB*, 1992.