Self-Organization in Computational Systems

Ricardo R. Gudwin – University of Campinas (UNICAMP), Campinas, Brazil

ABSTRACT: Computational systems (software) differ from material systems or living systems due to not having the materiality that usually grounds natural self-organizing systems. Nevertheless, in many computational systems (for example, intelligent systems) learning and adaptive abilities suggest that some sort of self-organization is possible. In normal software systems, the code structure is fixed, and even though data can change in time, the data structure is also fixed. But the situation can be different in adaptive and learning systems, where the whole data structure can adapt and change in time. In some situations even the code structure can change, giving the system the ability of self-production, which seems to be a mark of self-organization. This chapter presents an analysis of this kind of computational system in particular, and the author argues that self-organization is possible in computational systems.

Introduction

The overall definition of a *system* can be quite inclusive. According to (BRESCIANI & D'OTTAVIANO 2000), a system can be any unitary entity, from a complex and organized nature, constituted by a non-empty set of active elements, which maintain some relation to each other with time-invariant characteristics, giving them, as a whole, an identity. Or, in other words, a set of elements forming a structure with a given functionality, gaining an identity due to providing this functionality. Systems can be *natural*, as in the case of living systems, or *artificial*, as in the case of technological artifacts built by men. The *organization* of a system pertains to the structure connecting the many different system parts, meaning that each part is in some kind of relation to one or more other parts of the system. This connection might be physical, or simply logic, implying that there is some relation between two specific parts within the system. The term *organization* can be ethimologically traced back to the process leading to the formation of *organs* in a living body, i.e. a collection of parts of a physical body exhibiting collectively, some sort of functionality, and gaining together the status of an organ due to performing this functionality (the greek word *organon* means: *tool*). In a men-made technical system, the idea of *organization* might not be really related to true *organs*, as in a living body, but have to do with the existence of *sub-systems* of the overall system, carrying on some sort of functionality, and having on themselves some sort of identity as a composite of parts.

These definitions are definitively quite abstract, but can be applied to either living beings, technical artifacts or social entities like business organizations or groups of people. But it is important to point out that the term *organization* holds a double meaning. It might designate either a fixed structure defining the system (an atemporal structure or, in a temporal system, its particular structure in a given instance of time), which we will be calling here an s-organization (meaning an organization as a structure) or can be related to a dynamical process, where this structure is modified along time, which we will be calling here a p-organization (meaning an organization as a process). In other words, the term *organization* might be related either to the structure configurating a system in a given instant of time (as e.g., when we talk about the *current* organization of a system, an s-organization), or to the process where this structure might be changing over time (as e.g. when we talk about the process of organization of a business company department along history, a p-organization). This double meaning can be quite misleading, specially if we are talking about a specific phenomenon: self-organization.

According to (DEBRUN 2009a; 2009b), a system can be *hetero-organized*, if the system structure is imposed by an external source, or *self-organized*, if it produces itself, i.e., if it evolves by itself, as an offspring of an interaction of a set of parts, which were independent before they gain together its entity status (its identity as a system) due to providing a functionality. Debrun also differentiates between a *primary self-organization*, if the system gains its identity *during* the initial interaction of its many parts, before they can be seen as a system, and *secondary self-organization*, if the system, after its identity is already determined, is able to preserve its structure, if this structure is disturbed in any form.

Using our terminology, self-organization is achieved if a system might reach a stable s-organization due to its p-organization, or similarly, if it reaches its s-organization due to its p-organization. If this stable s-organization is achieved during the p-organization process, primary self-organization is achieved. If the system is originally hetero-organized, but is able to maintain its stable s-organization due to its p-organization, if its structure is disturbed in some way, we might say that the system reaches just secondary self-organization.

Usually, machines and other men-made systems are hetero-organized, as their structure don't change along time. This is the most common case for men-made systems. Converselly, natural systems like living beings can be seen as (primary) self-organized systems. It is important to point out, though, that a system not necessarily must have material parts. They might be just logical. An interesting case we want to investigate here, is the case of computational systems. Not any kind of computational systems, but specifically *software*. Our first guess on this kind of system is to necessarily classify them as hetero-organized, as usually software is originated from the work of a human programmer, which externally imposes to the software a fixed structure, which does not change and gives it its identity as a system, providing some sort of functionality. But we would like to investigate if some sort of self-organization is possible in software systems. And, if this self-organization is possible, would it be just a secondary self-organization, or a primary self-organization might also be possible ? [1]

Some Characteristics of Software Systems

As we already pointed out, software systems are intrisecally different from other kinds of systems, because of its lack of materiality. A software system is just pieces of data stored in a computer memory, being executed by a computer processor. This data might be stored in files, or be already loaded in the computer's memory. Even though there is some sort of materiality, because this data is stored in a physical memory, this is not what gives a software system its identity. If software is moved to a different kind of memory, it will be still the same software. So, software is an example of purelly logical systems. Let's start our analysis with some characteristics which are common to all software systems. Even though a piece of software is just data on memory, in fact, it is important to split this data in two different categories, which are functionally different. In any software system, part of the data is what we call *code*, i.e. instructions in processor language (machine language) which are to be executed by the processor during the software execution. The rest of the data is *data proper*, i.e., data contents which are meant to be used as variables in the software execution. They might contain some initial data which might be important for software execution, or be just a placeholder for the data which will be processed during software execution. Now, from a systemic point of view, we might define that *code* embeds the structure of the system, and *data proper* is just a set of states which describes the system inner working. In a traditional hetero-organized software system, the code is generated by a programmer, conceived usually in a high-level programming language and compiled into bytecodes in

1 It is important to mention here that even though we are referring to Debrun's notions of primary and secondary self-organization, we will be exploring here new possibilities which were not imagined by the time of his original proposals. Some authors might disagree that primary self-organization is possible in the cases discussed in the present work.

order to constitute the system structure. This structure never changes during system execution and the system provides its functionality, as conceived by design.

But things might become a little fuzzy if we start considering a special class of software systems: self-modifying systems. Because computer instructions can change the contents of memory, in general, and the code is stored into memory, it is possible to build software programs which are able to modify their own code. Usually, software systems are supposed not to do that, but it is feasible, and in fact it is done in some cases which we will be detailing in the sequence. If we consider that the code is the system structure, than, this opens a possibility for the system to changing its own structure and so opening the possibility for having self-organized software systems.

We would like also to point out some characteristics which are typical from two different kinds of software: *virtualizers* and *simulators*. *Virtualizers* are software programs which are used to be functionally equivalent of a whole computer running a different kind of system. Virtualizers are becoming very popular, as a way of testing multiple kinds of systems within the same host computer. In a so called *host system*, the virtualizer creates a virtual machine, functionally indistinguishable from a real machine (except for performance issues), where another system, called the *client system* can be installed and executed. Using a virtualizer, it is possible, e.g. to run a Linux system (as a client system) on top of a host system running MS-Windows. The user might think she is running a Linux machine, but in fact the real computer is running MS-Windows. But, besides issues related to computer performance (because a virtualized system runs slower while compared to its host system), the user might have a completely Linux experience, even though in the background there is really a MS-Windows system being executed. Virtualizers are important in our case, for us to understand that it is possible to run a completely different system, as a client system, virtualized on top of a host system. This idea is important for us, because we might conceive a host system, which is hetero-organized, virtualizing a client system that might be a self-organized system. *Simulators* are somewhat similar to virtualizers, with a distinction. In the case of a simulator, many properties of the real system being simulated are the same, both in the real system, and its simulation. But some of these properties are not. For example, a simulator can preview the amount of rain in a given location, along time, but the simulated rain is not wet, and do not have some other properties of real rain. Simulators are very important in engineering, because they are very effective in allowing predictions on the real system behavior, without the burden to have available a real system which might be expensive or time-consuming. Simulators are important for us because even though there might be properties of the real system which are not present in the simulated system, if we build the simulation with care, the most important properties (those we are interested in studying) can be brought into the simulation, and in this case, the simulated system embeds in itself everything which is important for us to derive conclusions. In other words, if we are building a simulation of a real process which seems to be self-organized, there is a chance that the simulated system is also self-organized.

A final concern must be addressed as well. A computer is inherently a deterministic machine, which means that given the same initial conditions, a computer program will always generate exactly the same behavior. Apparently (BONABEAU et.al. 1997), randomness is a requirement for self-organized systems[2] . If we want a computational system to be self-organized, how to conciliate that ? The issue on randomness is since a long time a concern in the construction of simulators (JAMES 1995; WANG 1996). The solution to this problem is in the use of pseudo-random number generators (HULL & DOBELL 1962), which from a given seed as input, typically exhibit statistical randomness while being

2 Even though Debrun didn't required true randomness, but only a randomness in the sense of Cournot, of having a statistical independence among the interacting parts, allowing new forms to emerge, not previously determined by a general law.

generated by an entirely deterministic causal process. Even though the sequence is deterministic, if we use a random seed, as e.g. the milisecond in which the simulation was started by the user, the sequence will be completely different, each time the program using it is run. And, from a statistical point of view, the sequence provides a behavior which can be classified as random.

Object-oriented Adaptive Software

Object-orientation is one important metaphor used for the constuction of software systems. In this metaphor, the software can be seen as a set of objects, interacting among themselves by means of message passaging. The user provides events, like clicking the keyboard or moving the mouse, and these events are sent to interface objects in the system, creating a cascade of messages, which will result in the system functionality. We can conceive an object-oriented program as a virtualized system on top of an operational system. The reaction to messages can be either deterministic, in some cases, or random, in other cases, using pseudo-random generators. Objects might have their internal variables changed, and either create new objects or destroy other objects during its behavior. In principle, object-oriented systems are supposed to be hetero-organized, because they need to be developed by a computer programmer. But supposing a process of virtualization, with the aid of pseudo-random number generators, it is possible to conceive that a self-organized client system can evolve on top of a hetero-organized host system. Instead of working with a pre-designed set of objects, we might think of some sort of adaptive object-oriented system, where new objects can be created and the possible interaction between objects can be governed by random number generators. In the same way in a physical system a set of physical rules govern the possibilities of physical objects to interact, paving the way for the evolution of self-organized systems, a given set of rules can be programmed in object-oriented systems, and with the aid of randomness brought by pseudo-random number generators, self-organized object-oriented systems can be conceived, with a bonus: in material systems, the rules are those that exist. In an object-oriented adaptive software systems, these rules can be changed and tested, letting different rules govern the dynamics among software objects. Hetero-organized rules work just like physical laws in the material world. The resulting system, though, can be self-organized.

Both primary self-organization and secondary self-organization can be possible. In the case of secondary self-organization, an initial structure (an initial configuration of objects) is hetero-organized, and put to interact to each other. After that, this initial structure might evolve along time, changing its structure, but maintaing its identity. But even primary self-organization is possible. We might have a completely random generation of objects, and random possibilities of interaction between these objects, as is common in a special kind of systems called evolutionary systems.

We provide a better understanding of some examples in the next sections.

Learning Software Systems

Let's start our investigation of this possibility of finding self-organization in software systems with a particular kind of software: Learning Systems. Learning Systems started to appear from within the context of Artificial Intelligence, and nowadays are becoming very popular, gaining its own field of research: Machine Learning. A typical learning system is what we call a *neural network*. A neural network comprises a network of entities called artificial neurons, which are abstractions of real neurons as in the brain of living creatures. In some sense, they are a kind of a simulation of a network of real neurons. The artificial neuron does not have all the properties of a real neuron, but they share with them many properties, which might turn an artificial neural network into a self-organized system.

A neural network typically exhibits only secondary self-organization. This happens because usually, the initial structure of a neural network is hetero-organized, and after the neural network starts operating, it changes this structure in order to adapt and learn. But there are specific neural networks, called constructive neural networks, which usually start with just one neuron, and new neurons are created and incorporated to the network, as it interacts with its environment. Such constructive neural network appears to have all the conditions for being classified as primary self-organized systems.

Evolutionary Systems

Another kind of system which is worth to mention here is the class of *Evolutionary Systems*. Evolutionary systems are a kind of intelligent systems where many aspects of biological evolution are simulated in a computational environment. Despite its many variations, evolutionary systems usually comprise a population of elements, which are processed in an iterative way, using combinatory operators like crossover, mutation and others, generating new populations along time. Each element in the population represents the solution of a mathematical problem, in the form of a structured collection of parameters, which are important for characterizing the mathematical problem. This element is usually called a chromossome, in a straight analogy to biological evolution, which can be evaluated by a *fitness function*, providing a measuring for the quality of the solution brought by a particular element. With this, in a given population many different possible solutions can be evaluated, and usually the best solution is considered as an output from the evolutionary system. The initial population is usually randomly generated, and as soon as new populations are generated through this evolutionary process, solutions of a better quality are derived (or "evolved"), making the evolutionary system a kind of optimization process. Each evolutionary step implies into a first step, where the population first grows, using the many combinatory process (like crossover, mutation, etc), and later contracts, when a selection process maintains in the final population, only the elements considered more apt, i.e. those with a greater fitness value. With this expansion/contraction movement, the size of a population can be maintained constant along the generation of successive populations.

Evolutionary systems, as in the case of neural networks, can be classified as self-organized systems. We might identify either primary and secondary aspects of self-organization in their functioning. As populations are initially randomly generated (using, of course, pseudo-random number generators in their implementation in computers), they might require many steps of evolution before reaching their final configuration, where a good solution for a mathematical problem is generated. This characterizes its primary self-organization properties. But then, after good solutions are available in the population, the evolutionary system preserve those of a better quality in the next populations, meeting the criteria for being classified also as a secondary self-organized process.

Discussion

We presented some examples of computational systems where apparently some sort of self-organization process seems to be operational. As we pointed out, the first impression would be that true self-organized processes might be impossible in computational systems. This impression comes from the fact that computational systems are programs stored in a computer memory, and if these programs are generated by a human programmer, they have a fixed structure and could not be classified as self-organized systems. More than this, these programs lack some kind of materiality, which is usual in more common kinds of self-organized systems, like biological systems. A third argument against the possibility of self-organized systems in computational systems is the lack of true randomness, as

computers are deterministic machines, where a system with the same set of inputs will always exhibit the same kind of behavior as output.

But then we remember that the definition of *system* implies in it being just an abstraction for a fragment of reality. This opens up the perspective of a system as being an abstraction for some generic sort of support, being this support a fragment of reality, or another system, i.e., another abstraction. This new perspective allows us to understand the notion of virtualization among systems, where multiple layers of virtualization are possible until we ground them on a fragment of reality. Having these multiple layers in mind allows us now to speculate on the possibility of having self-organized systems virtualized on top of hetero-organized systems. And this is the key for us to understand how self-organized systems are possible to happen in computational systems. Even though the host system is hetero-organized, this host system embedds another client system, which is virtualized or simulated, and this virtualized/simulated system can be self-organized.

There is still the problem of true randomness. But then, we remember that with the use of pseudo-random number generators, and with a true random seed, e.g. the precise milisecond where the program is started, the pseudo-random number generator can generate an unique sequence of numbers which are in principle indistinguishable from a true random sequence. This leads us to speculate, first, if randomness really exhists in the natural world, or are they just sequences with the same property as those generated by a pseudo-random number generator ? Secondly, are we really sure that true randomness is a strong requirement for self-organization ? Or the properties given by pseudo-random number generators are sufficient for self-organization to appear ?

Apparently, some kinds of software systems, like object-oriented adaptive software, learning systems and evolutionary systems, in being considered as virtualized/simulated systems on top of some sort of hetero-organized support system, might hold all the requirements for being considered self-organized. They might have a (pseudo) random s-organization (i.e. a random initial stucture) and a set of laws governing their behavior, provided by the support hetero-organized system, which are equivalent to the physical laws governing a material system. This s-organization have means for changing itself over time, implementing a p-organization that might conduce this s-organization to the formation of stable units, giving them identity through providing them some functionality. This p-organization can also maintain this s-organization stable, if it is disturbed by some external inputs, giving them the properties required for secondary self-organization. In summary, they apparently hold all the requirements for being classified as self-organized, either primary in some cases, or just secondary in other cases. The hetero-organization provided by human programmers to this software, can be comparable to the fixedness of natural laws imposed to material entities in a biological self-organized system, i.e., they can be considered as just "given", with the difference that in the natural world these laws are unique and cannot be changed, while in the computational world, we are able to explore alternative realities, something which is not possible in the case of the material world.

Conclusions

In conclusion, we propose that some kinds of computational systems, e.g. adaptive object-oriented systems, learning systems and evolutionary systems, hold all the requirements for being classified as self-organized systems. The argumentation provided in this text does not completely prove this is the case, but brings strong evidence that they hold the requirements for being classified like that.

It is important to point out, though, that further steps are still necessary to be investigated. First, if our hypothesis that pseudo-random number generators are indeed enough for self-organization to appear, or

there might be some property of such pseudo-number sequences which are absent and preventing true self-organization to evolve. It will be important to point out, though, what is this property and why it is preventing true self-organization to appear, if it is the case. Second, we must explicitly identify what are the specific conditions of these systems which enable them to become self-organized.

Hopefully, with the continuation of this investigation, we will be able to have a better comprehension of what is self-organization, and what are the minimum requirements for a system, in order to be classified like that.

References

BONABEAU, E., THERAULAZ, G., DENEUBOURG, J. L., ARON, S., & CAMAZINE, S. (1997). Self-organization in social insects. *Trends in Ecology & Evolution*, *12*(5), 188-193.

BRESCIANI F., E. ; D´OTTAVIANO, I. M. L. . CONCEITOS BÁSICOS DE SISTÊMICA. In: I.M.L.D'Ottaviano; M.E.Q.Gonzales. (Org.). Auto-Organização Estudos Interdisciplinares. Campinas,SP: UNICAMP-CLE, 2000, v. 30, p. 283-306.

DEBRUN, M. . The Idea of Self Organization. In: I.M. L. D'Ottaviano; M.E.Q. Gonzales (orgs). Michel Debrun - Brazilian National Identity and Self Organization. Campinas, SP: UNICAMP-CLE, 2009a, v. 53, p. 31-52, reproduced in this volume as chapter 1.

DEBRUN, M. . The Dynamics of Primary Self Organization. In: I.M. L. D'Ottaviano; M.E.Q. Gonzales (orgs). Michel Debrun - Brazilian National Identity and Self Organization. Campinas, SP: UNICAMP-CLE, 2009b, v. 53, p. 75-110, reproduced in this volume as chapter 2.

D'OTTAVIANO, I.M.L. and BRESCIANI F., E. (2018) Basic Concepts for a General Theory of Self-Organizing Systems - this volume, chapter 3

HULL, T.E. and DOBELL, A.R., 1962. Random number generators. *SIAM review*, *4*(3), pp.230-254.

JAMES, F. "Chaos and Randomness." *Chaos, Solitons & Fractals* 6 (1995): 221-226.

WANG, Y. - Randomness and Complexity, PhD thesis, University of Heidelberg, 1996.