

# Emergence as a General Architecture for Distributed Autonomic Computing

*Tom De Wolf*  
*Tom Holvoet*

*Report CW 384, January 2004*



Katholieke Universiteit Leuven  
Department of Computer Science  
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

# Emergence as a General Architecture for Distributed Autonomic Computing

*Tom De Wolf*  
*Tom Holvoet*

*Report CW 384, January 2004*

Department of Computer Science, K.U.Leuven

## **Abstract**

Today's systems are becoming more and more complex, i.e. distributed, situated, open, and dynamic. Autonomic computing aims to deal with the complexity autonomously. Hence, distributed autonomic computing systems tend to consist out of autonomous entities because of the increased distribution. This increased complexity and autonomy makes it difficult to build systems with a global coherent behaviour as a huge number of entities are expected to cooperate. There are signs of a future for emergence in multi-agent systems as a general architecture for the individual entities and the distributed autonomic computing system as a whole. Understanding and exploiting the process of emergence is key to study and engineer global coherent behaviour, and thus key to develop and manage distributed autonomic computing systems efficiently. As a consequence, there is a need for fundamental scientific methods to gain those insights and control tomorrows emergent autonomic computing systems. Chaotic time series analysis is one scientific method that can provide those fundamental insights.

**Keywords :** emergence, distributed autonomic computing, chaos theory .  
**AMS(MOS) Classification :** Primary : D.2.0, A.1. Secondary : G.3, I.2.11, C.2.4, C.5.0.

# Emergence as a General Architecture for Distributed Autonomic Computing

Tom De Wolf and Tom Holvoet  
DistriNet, Department of Computer Science, KULeuven  
AgentWise Lab  
Celestijnenlaan 200A, 3001 Leuven, Belgium  
{Tom.DeWolf, Tom.Holvoet}@cs.kuleuven.ac.be

## Abstract

*Today's systems are becoming more and more complex, i.e. distributed, situated, open, and dynamic. Autonomic computing aims to deal with the complexity autonomously. Hence, distributed autonomic computing systems tend to consist out of autonomous entities because of the increased distribution. This increased complexity and autonomy makes it difficult to build systems with a global coherent behaviour as a huge number of entities are expected to cooperate. There are signs of a future for emergence in multi-agent systems as a general architecture for the individual entities and the distributed autonomic computing system as a whole. Understanding and exploiting the process of emergence is key to study and engineer global coherent behaviour, and thus key to develop and manage distributed autonomic computing systems efficiently. As a consequence, there is a need for fundamental scientific methods to gain those insights and control tomorrows emergent autonomic computing systems. Chaotic time series analysis is one scientific method that can provide those fundamental insights.*

## 1. Introduction

An important fact is that current systems are becoming more and more complex, i.e. distributed, open, dynamic and a high degree of locality. Examples are telecommunication networks, flexible manufacturing networks, urban traffic networks, economic systems such as a supply chain, and simulation of ecological systems.

On top of that evolution, we want to build more and more autonomic computing systems. The paradigm of *autonomic computing* aims to design and build computing systems capable of adjusting to varying circumstances by taking the initiative to decide and do things themselves. Autonomic computing systems need to anticipate needs autonomously. The real complexity is hidden from the user. In the autonomic computing perspective this is described by the goal to

achieve the self-X principles (see [10]). Users can concentrate on *what* they want to accomplish rather than figure out *how* to manipulate the computing systems to get the wanted behaviour.

The increase in distribution, openness, situatedness and dynamics means that systems need to handle more and more complexity on their own. Also, making increasingly distributed systems autonomous implies using individual autonomous entities inside the system. The ultimate challenge is to find approaches that can help us understand the dynamics and mechanisms that can control the global behaviour of these distributed autonomic computing systems. This paper describes *emergence* in multi-agent systems as a promising approach that matches this profile.

To use emergence as a general architecture for distributed autonomic systems it is important to understand how emergence works and which mechanisms enable emergence. This understanding is key to conquer the difficulties with building emergent systems. *Fundamental science* is essential to gain insights about emergence. In this paper we propose chaotic time series analysis as one scientific method that can provide some of those fundamental insights.

In the next section, we describe what emergence is. Then, in section 3, we state why emergence is important in relation to distributed autonomic computing. In section 4, the need for fundamental science towards understanding and controlling emergent systems is described and as an example the use of chaotic time series analysis is given. The paper ends with a conclusion and some future work.

## 2. Defining Emergence

There is a lot of confusion in literature about the exact definition of emergence [8, 2, 4, 5, 6, 15, 16, 18]. Therefore, without stating that the following definition is the only correct definition of emergence, we use it as our working definition:

“Emergence is a dynamic nonlinear process that leads to emergents (properties, behaviour, structure, patterns, ...) at the macro-level of a system from the (inter)action of the parts at the micro-level. Such emergents are novel, i.e. they can not be readily understood by taking the system apart and looking at the parts (= reductionism). They can however be studied by looking at each of the parts in the context of the system as a whole”

The ‘level’ mentioned in the definition refers to certain points of view to look at the system. The macro-level considers the system as a whole and the micro-level considers the system from the point of view of the individual entities that make up the system.

We give a simple example from biology to clarify this, an ant colony. At the micro-level each ant observes its local environment and performs local actions accordingly. The macro-level is looking at the system as a whole. Thus, for example, path formation with pheromones is an emergent that is situated at the macro-level of the system. When we look at a single ant, it has no clue about the existence of paths at all.

Some important characteristics of emergence are the following:

- **Micro-macro effect [8, 2, 4, 5, 6, 15, 16, 18]:** this is the most important characteristic and is mentioned explicitly in most definitions of emergence in literature, it was explained above.
- **Radical novelty [2, 5, 4, 1]:** this refers to the non-reductionism to the individual entities. The global behaviour is novel w.r.t. the individual behaviours at the micro-level.
- **Coherence [4, 16, 15]:** emergents appear as integrated wholes that tend to maintain some sense of identity over time (i.e. a persistent pattern). Coherence spans and correlates the separate lower level components into a higher level unity.
- **Dynamical [4, 6]:** emergents arise as a complex system evolves in time. Emergent behaviour is also very dynamic because at the micro-level there must be a constant dynamic to maintain the global behaviour.
- **Nonlinearity [4, 6]:** emergence requires the “small cause, large effect” principle and has an intense focus on nonlinear interactivity. It is this nonlinearity that makes it possible to get those secondary effects at the macro-level that we call emergents.
- **Distributed Control [15, 6]:** you control the action of the parts, not the whole.
- **Interacting parts [15]:** the parts must be interacting - parallelism is not enough. Without interactions, interesting macro-level behaviours will never arise.
- **Robustness and Flexibility [15, 6]:** emergents are relatively insensitive to perturbations or errors, and have a strong capacity to restore themselves. Increasing damage will decrease performance, but the degradation will be ‘graceful’.

These characteristics are very relevant in distributed autonomic computing systems. We elaborate on this in the next section.

### 3. The Importance of Emergence

Several characteristics distinguish distributed autonomic computing systems from ‘traditional’ ones. Examples in different areas show that these characteristics influence many application domains. In [25] this is described as a sign for a revolutionary shift of paradigm, and likely to change our very attitudes in software systems modelling and engineering. In what follows, we describe those distinguishing characteristics and relate them to the concept of emergence in multi-agent systems that was described in the previous section.

#### 3.1. Characteristics of Distributed Autonomic Computing systems

In this section we describe characteristics (partially inspired by [25]) of distributed autonomic computing systems that will become even stronger when we evolve to completely distributed and completely autonomic systems.

- **Distribution:** systems become more and more distributed. This includes physical distribution, due to the invasion of networks in every system, and logical distribution, because there is more and more interaction between applications on a single system and between entities inside a single application.
- **Situatedness:** systems become more and more situated: there is an explicit notion of the environment in which the system and entities of the system exist and execute, environmental characteristics affect their execution, and they often explicitly interact with that environment. Such an (execution) environment becomes a primary abstraction that can have its own dynamics, independent of the intrinsic dynamics of the system and its entities. As a consequence, we must be able to cope with uncertainty and unpredictability when building systems that interact with their environment. This situatedness often implies that only local information

is available for the entities in the system or the system itself as part of a group of systems.

- **Openness:** being situated in an environment, perceiving it, and being affected by it intrinsically implies a fuzzy boundary between the system and its environment. The system is no longer isolated, but becomes a permeable subsystem, whose boundaries permit reciprocal side-effects that are often extreme and complex.

This openness also results in structural dynamics, i.e. dynamic changes in system structure (systems or entities in a system can move between environments, interaction channels change constantly and thus interacting with the same entity is unlikely,...). This structural dynamics makes it even harder to understand and control the overall behaviour of a group of systems or a single system.

- **Locality in control:** the 'flow of control' concept has always been one of the key concepts of computer science and software engineering, at all levels, from the hardware level up to the high-level design of applications. However, when software systems and components live and interact in an open world, the concept of global flow of control becomes meaningless.

Independent software systems have their own autonomous flows of control, and their mutual interactions do not imply any join of these flows. The concept of 'global execution control' disappears. This trend is made stronger by the fact that not only do independent systems have their own flow of control, but also different entities in a system have their own flow of control.

- **Locality in interaction:** when we have a physical environment, locality of interactions is automatically enforced by physical laws. In a logical environment, if we want to minimise the conceptual and management complexity we must also favour modelling the system in local terms and limiting the effect of a single entity on the environment. Locality in interaction is a strong requirement when the number of entities in a system increases, or as the dimension of the distribution scale increases.

In any case, tracking and controlling concurrent and autonomously initiated interactions is much more difficult than in object-oriented and component-based applications, even if these interactions are strictly local. The reason for this is that autonomously initiated interactions imply that we can not know what kind of interaction is done and we have no clue about when a (specific) interaction is initiated.

- **Dynamic:** as pointed out a number of times in the previous characteristics, systems are becoming more and more dynamic in a number of aspects such as dynamics from the environment, structural dynamics, huge interaction dynamics and from a software engineering perspective the rapidly changing requirements for the system. For example, in manufacturing systems, the competitive pressures are reducing product life times and thus forcing systems to change frequently. Also, machine failures and upgrades force the system to adapt to these changes. In such a situation, the system needs to be very flexible and dynamic.

- **Autonomy:** as a direct consequence of the autonomic computing perspective, systems need to incorporate more and more autonomy to eventually reach the goal of handling all complexity autonomously.

- **Need for global behaviour:** the characteristics described so far, make it difficult to understand and control the global behaviour of the system or a group of systems. Still, there is a need for a coherent global behaviour. Some (non-)functional requirements that have to be solved by computer systems are so complex that a single entity can not provide it. We need systems consisting out of multiple entities which are relatively simple and where the global behaviour of that system provides the functionality for the complex task.

This need is also present in traditional object-oriented systems but in that case the presence of a single flow of control makes it possible to have global execution control and thus control the global behaviour of that system. When the entities are more than objects, i.e. they become autonomous, that flow of control stays local in each entity and thus a global execution control is not feasible.

### 3.2. Why using Emergence and Multi-Agent Systems?

Now that we have given a number of characteristics of distributed autonomic computing systems, we state in this section that engineering such systems based on emergence in multi-agent systems and exploiting the advantages of emergent behaviour is a promising approach to handle the complexity of those autonomic computing systems and this complexity will only increase in the future.

**Matching Characteristics.** First of all, there are a number of characteristics of emergence and distributed autonomic computing systems that match. To be able to focus on emergence we need distributed control and interacting parts. Both are intrinsic characteristics of distributed autonomic computing systems. Emergence is dynamical in the sense that emergents arise over time and dynamics at the

micro-level are needed to maintain the emergents. The increasing dynamics of distributed autonomic computing systems support this and emergence itself makes it possible to cope with the dynamics that occurs in those autonomic computing systems. The latter is made clear below. The presence of intrinsic *distribution, interacting parts* and *dynamics* are basic ingredients for exploiting emergence.

Secondly, emergents are relatively insensitive to perturbations or errors. They are very *robust and flexible*. This flexibility supports the dynamical characteristic of the systems we consider. The environmental dynamics, structural dynamics and rapidly changing requirements and the uncertainty accompanying it require that the behaviour of the system is robust and flexible.

In engineering distributed autonomic systems we want a *coherent global behaviour*. Coherence is a characteristic of emergence. Thus, engineering the global behaviour as emergent implies coherence.

**Advantages.** Using emergence has a number of advantages. The radical novelty and micro-macro effect of emergence implies that not one entity has to be aware of that global behaviour. This means that no global and central planning of the global behaviour is necessary.

Because the global behaviour is not present in each entity, such an entity can be kept as simple as possible. Even in a system containing only very simple rule-based entities a complex emergent behaviour can arise. This simplicity favours engineers that have to build such systems. To manage a single entity becomes easy. However, managing the macro-level is not that easy (see section 4).

An entity is not a single point of failure. This again, refers to the advantageous characteristic of emergence, namely robustness and flexibility of the system. Failure only manifests itself in graceful degradation.

Emergent behaviour is not only a coherent global behaviour that persists over time and maintains some identity over time. Because of the dynamics at the micro-level, a system based on emergence is also very flexible and adaptive in adjusting its behaviour.

**Why do we need emergence?** The greatest difficulty in engineering distributed autonomic systems is to get a coherent global behaviour as mentioned above. If we want such a coherent global behaviour in more and more distributed, situated and autonomous systems, we can build this global behaviour as an emergent property of the system. Coherence is a characteristic of emergence. If we would not use emergence to handle this at least one entity in the system must have a model of the wanted global behaviour and coordinate the whole system. This implies central control in the worst case scenario and is not feasible in increasingly complex systems where the number of entities that act autonomously in the system is very large. The characteristics of locality in control, interaction and information contra-

dict with this approach and supports emergence where not one entity needs global information, control or interaction.

**What to do when engineering emergence?** An essential characteristic of emergence, that is not necessarily present in distributed autonomic computing systems, is non-linearity. We must incorporate nonlinearity in engineering emergent autonomic computing systems by using nonlinear mechanisms that result in the wanted behaviour.

**Multi-agent systems as technology.** When we combine the characteristic of autonomy with the increased distribution, situatedness, and locality of control and interaction, building a completely autonomic system can only be done when the entities inside that system also incorporate autonomy as one of their characteristics. Multi-agent systems play an important role as a flexible technology that envisions to build systems consisting out of a number of autonomous entities, called agents.

**Conclusion.** In conclusion, we can state that getting the problem solving power from emergent phenomena in multi-agent systems supports the needs and challenges we will face when engineering distributed autonomic computing systems.

## 4. Difficulties of Emergence

The idea of emergence is clear but building systems that exhibit the emergent behaviour that matches the requirements has a number of difficulties:

- Emergence is a non-linear phenomenon and the lack of experience in using nonlinearity in engineering systems today means that using emergence can not rely on a structured methodology to use.
- Individual entities must cope with a lot of uncertainty such as not knowing which other entities it will interact with, unanticipated dynamics of the environment in which they live, and incomplete data and knowledge in general due to the increased locality. This uncertainty makes it difficult to build individual entities so that they can react or not react to a certain dynamic in a way that preserves the required global behaviour.
- Often emergence is only seen if the amount of entities in the system is large enough. If 'large enough' refers to a huge number so that the performance and efficiency of the system degrades enormously due to a combinatorial explosion of states, using emergence becomes problematic. But, maybe using the right nonlinear mechanisms limits the entities needed.
- When building a system we have an idea about what the global behaviour should be. How can we decompose that behaviour and make sure we give the entities the correct behaviour so that the resulting global

behaviour matches the requirements? There are no guidelines to help with this yet.

It is obvious that the main problem here is the current lack of understanding and insights about the process of emergence and the mechanisms that allow such behaviour to arise. In the next section we argue that it is important to have fundamental scientific methods to gain those insights.

## 5. Chaotic Time Series Analysis as a Fundamental Scientific Method

If we want to exploit emergence as a general architecture to engineer autonomic computing systems we must first gain a profound understanding into the dynamics of mechanisms that result in emergence. If we do not have this understanding, building distributed autonomic computing systems based on emergence would be stuck in a mere trial-and-error technique. To gain those insights there is a need for fundamental scientific methods that allow to analyse the behaviour of emergent systems. When we have certain insights into the process of emergence, the next step is to apply this when engineering autonomic systems.

We propose chaotic time series analysis as one such possible scientific method that can also be used to analyse autonomic computing systems in general[3]. The first part of this section describes what chaotic time series analysis is and offers. The second part discusses how we can use chaotic time series analysis to better understand emergence. In the last part of this section we describe what we can do with those insights to engineer self-X behaviour in distributed autonomic computing systems.

### 5.1. What is Chaotic Time Series Analysis?

Analysing the dynamics of systems enables to make predictions about the global behaviour, and gather insights into the effects of certain parts or even a single variable on the global behaviour of the system. Today we often make use of simple measures (e.g. averages over time) to characterise the behaviour of a system. Although useful characterisations for static systems, they are less helpful if the system is nonlinear and constantly subjected to change [23]. We need more advanced analysis techniques.

A lot of tools and techniques are available from chaos theory that was developed to describe nonlinear behaviour. To analyse the dynamics of today's complex systems and emergence, chaos theory offers [22]: a conceptual framework with which dynamical behaviour can be characterised, experimental techniques that can estimate these characteristics from time series of state variables in an observed system, and a good framework to explain principles of the (emergent) behaviour. In this section we first give a small

introduction into the concepts of chaos theory, followed by an overview of some analysis techniques.

#### 5.1.1 Concepts of Chaos Theory

Important concepts in chaos theory are the system state space, the trajectory and the attractor. The *state space* of a dynamical system is a collection of time-dependent variables that capture the behaviour of the system over time. The concept of a *trajectory* refers to the system's trajectory through state space, i.e. the successive values of state variables viewed as coordinates in state space. In many dynamical systems, the trajectory eventually enters a subspace which it never leaves. This subspace exerts an attractive force that draws the system into it, a so-called *attractor*. Several disjoint attractors can be present in a system with each its set of initial conditions from which trajectories enter a given attractor, called that attractor's *basin of attraction*. [17] gives a short introduction into these concepts.

#### 5.1.2 Analysis with Chaos Theory

The next step is knowing how to apply chaos theory. The techniques we address here are chaotic, nonlinear time series analysis techniques. This includes two parts: we need to identify the data we can measure and we need to analyse/interpret that data.

**What Data to Measure?** First of all we have to gather data from the system. The challenge here is to select the right data from the abundance of information that is available. To be able to do this we need some idea of what different kinds of measurements can tell us about the behaviour. Until now, there is no systematic approach to get to know this and exploration of possible measures is the only way to learn what sorts of information will be most valuable.

The most useful data is time dependent data because behavioural dynamics is always considered as evolving over time. This results in time series that can be analysed. For example, in a transport module of a factory you can measure the evolution of the transit times between two sensors to detect congestion patterns. These patterns may not be detected in inter-arrival times at a single sensor [23, 24]. Another example is found in [20]. In analysing the dynamics of a supply chain you can measure the evolution of the amount of orders placed at a certain factory or site in the chain. The inventory evolution is also useful.

These are all very application specific results and future exploration may lead to the discovery of some general guidelines to choose the right values.

**How to Interpret the Data?** After gathering the data we can analyse it. Chaos theory offers a number of analysis

techniques of which we give a few:

- Trajectories in state space can be used to find the attractors. Different types of dynamics lead to different attractor geometries, including point attractors (for systems that converge to a static state), limit cycles (for systems with periodic behaviour), and fractal structures ('strange attractors', characteristic of formally chaotic systems). In [17] a comparison is made between these kind of attractors and classes of systems. A divergent trajectory or no attractor corresponds to a system out of control. A point attractor is related to a goal oriented system. A limit cycle and strange attractor involve going concerns, a system that has to maintain a certain goal behaviour.
- System attractors (if sufficiently low dimension) can be reconstructed with time-delay plots (provided that the available data is sufficiently rich and accurate). In these plots each value in a time series is plotted on the y-axis against a previous value on the x-axis. These time-delay plots can be used to find distinguishing patterns of behaviour and this has been done in manufacturing [23, 24] and supply chains [20, 19, 21].
- Computing Lyapunov exponents of attractors can be done from time series. These exponents quantify both the strength of attraction exerted by the attractor on nearby points in the state space and the degree to which neighbouring points within the attractor diverge from one another (degree of chaotic behaviour). The Lyapunov exponent can thus be a stability measure for the system [13]. The exponent is for example used in [14] to find out how chaotic the behaviour of a robotic system is. Different control algorithms for the robot can then be compared based on this measure of chaotic behaviour.
- A recurrence plot is a two-dimensional representation technique that brings out deterministic dependencies among successive elements in a time series. They can for example instantly make it apparent whether a system is periodic or chaotic.

In controlling nonlinear behaviour, it is important to find tools to identify those sensitive regions where the state of the system is extremely vulnerable to external perturbations. A recurrence plot is a visualisation tool that provides a good global estimate of the distribution of these sensitive parameters. It allows to anticipate in which states of the system it might be very sensitive to small fluctuations and in which domains it is relatively robust and insensitive and also where attempts to control the system would be most promising[12].

But again, we need to learn which techniques are most appropriate. Applying these methods to characterise systems

in terms of its formal dynamics is not enough. Besides knowing that the system has an attractor of a certain shape or dimensionality we also need to correlate particular dynamical patterns with domain-specific behaviour. This can allow us to find out where there are problems or opportunities to control the behaviour of that system. The mapping on classes of systems in [17], described above, is a first step in that direction, but making more domain-specific and detailed interpretations needs to be done.

A first example of such an interpretation is found in [21] for supply chains: after some time, sites further in the supply chain are more likely to follow a large order with another large one, and a small order with another small one. In other words, their orders have become correlated in time. This was found from interpreting a stretched pattern around the  $x=y$  line in time-delay plots for order time series.

A second example is found in [23]. Through analysis of transit times with time-delay plots they find the following patterns: squares correspond to line stoppages, imperfect squares correspond to interacting stoppages, a distinct diagonal band in which transit times vary in small steps corresponds to dynamic congestion.

## 5.2. Analysing Emergence with Chaos Theory

Now we can look at how we can apply chaos theory to the process of emergence in order to understand its dynamics. First we argue why chaos theory and especially chaotic time series analysis is a promising approach. In the second part we give some ideas on how we can apply the analysis methods to emergence.

### 5.2.1 Why Chaos Theory for Emergence?

Because emergence is a nonlinear process and chaos theory focuses on this, using chaos theory is a relevant approach. But understanding the dynamics of complex emergent systems by reasoning analytically from the detailed internal structure of the system is not feasible. Analytic solutions are unavailable for systems beyond even a very modest level of complexity. The alternative approach is to study systems that are executing and to attempt to generalise from their behaviour. And that is why chaotic time series analysis is a promising approach to apply chaos theory to emergent automatic computing systems.

Using chaos theory as a conceptual framework means that the development of chaos theory in a specific problem domain such as engineering emergent systems can provide a new vocabulary for understanding what happens in that system and new technologies to manage and control those systems. For example, in [11] this is done for the domain of manufacturing systems.

### 5.2.2 How to Apply Chaos Theory to Emergence?

In this section we give some ideas that exemplify how applying chaotic time series analysis to emergence can give insights.

Chaos theory allows us to characterise the type of attractor of emergence. A behaviour that maintains a certain goal corresponds to a limit cycle or strange attractor type. Verifying the latter could give us the understanding that emergence, which is a goal maintaining behaviour, corresponds to a limit cycle or strange attractor.

In the same way, for example, we could use the Lyapunov exponent to actually measure the difference between the micro-level behaviour and the macro-level behaviour. For example, [18] indicates that the micro-level behaviour is less stable (less order) than the macro-level behaviour (more order). This should result in a larger Lyapunov exponent for the micro-level than for the macro-level. A larger exponent indicates a larger degree of divergence and thus a less stable behaviour.

As a last example, using recurrence plots to identify those regions in state space where the emergent behaviour is most sensitive to external perturbations can allow us to understand when and where we can try to control emergence.

### 5.3. Engineering Distributed Autonomic Computing Systems

After gaining insights from analysis we must control distributed autonomic computing systems to get the wanted emergent behaviour. A large majority of control applications implemented today fall into the category of centralised control. Of course, this is not feasible for the autonomic systems that are highly distributed. To be successful, the structure of the system has to be exploited: locality of control, interaction, distribution, ... . Emergence exploits this structure and gives additional advantages.

The results of the analysis done with chaos theory are exploited to control the system. Understanding emergence allows to control the system more effectively and efficiently. It can give us engineering principles to guide implementations for optimal control and to achieve the wanted behaviour. Many dynamical systems exhibit different behaviours, depending on the values of certain critical parameters. It can be expected that the analysis shows which parameters and influence points are critical for a specific system and when they exhibit which emergent behaviour. Once we know the critical parameter to influence we must find out with which control mechanism we can influence that parameter.

In [20] this is applied to supply chain management and the authors show that the consumer can exert control on the global behaviour of the supply chain through the line of orders it issues. The magnitude of these orders can determine

to which point attractor or limit cycle the system is drawn, i.e. which global behaviour the system exhibits.

To control emergent behaviour (i.e. maintaining a certain behaviour), entities of a system can influence that global behaviour by manipulating simple, but critical parameters in their local environment. Thus, *local actions* can influence the global behaviour in a dramatic way and results from chaotic time series analysis are needed to know where and which local actions are useful. For example, in [24] undesirable emergent behaviour is damped out by using micro-controllers, acting as agents, that monitor their environment and take local action.

Local actions and the insights from recurrence plot analysis on sensitive regions can be used together to control the emergence. In autonomic systems this control can then be done autonomously by letting the entity in the system react with the right local action when the sensitive region is detected.

Also the *stabilisation* of the emergent behaviour is a control that can be enforced. We mentioned the Lyapunov exponent as a measure for stability. It has already been used to control the stability of robotic behaviour [14]. Other examples that control stability are [7] and [9]. Here, more diversity in the units that compose the system leads to more stability. This diversity emerged out of a homogeneous group by rewarding agents according to their actual performance and this leads to elimination of chaos.

The idea to measure the stability of the micro and macro level separately can confirm that less order at the micro level can give more order at the macro level. This insight can be exploited in engineering emergence because then we know that putting too much order in the behaviours of the individual entities is not such a good idea to get ordered emergent behaviour.

## 6. Conclusion

This paper shows that there are signs of a promising future for emergence as a general architecture for distributed autonomic computing systems. To summarise this we can state three things.

First of all, *computing systems are becoming more and more complex* because of increasing distribution, interconnection, situatedness, openness and dynamics. Next to this increasing complexity, in the autonomic computing perspective, systems will become more and more autonomous in handling that complexity. This makes it hard to build such systems with a coherent global behaviour.

Secondly, using the novel paradigm of *emergence in multi-agent systems as a general architecture* for the individual components and the distributed autonomic computing system as a whole opens possibilities to exploit these

complexity increasing characteristics and reach a global coherent behaviour.

Thirdly, because of current difficulties with using emergence, we must understand how it works and how we can exploit its characteristics in engineering such systems. This implies a *need for fundamental scientific methods to gain insights* and eventually control tomorrow's emergent autonomous computing systems. Chaotic time series analysis was given as a promising candidate to achieve this.

Future work will include studying and using chaotic time series analysis and other fundamental analysis techniques to analyse emergent systems and extract insights and guidelines to engineer such systems. Based on these insights and guidelines a case study can be developed that exemplifies the use of emergence in distributed autonomous computing.

## References

- [1] Y. Bar-Yam. *Dynamics of Complex Systems*, chapter 0, Overview: The Dynamics of Complex Systems - examples, questions, methods and concepts. Studies in Nonlinearity. Westview Press, July 1997.
- [2] J. Crutchfield. Is anything ever new? considering emergence. Working Paper 94-03-011, Santa Fe Institute, 1994.
- [3] T. De Wolf and T. Holvoet. Towards Autonomous Computing: agent-based modelling, dynamical systems analysis, and decentralised control. In *Proceedings of the First International Workshop on Autonomous Computing Principles and Architectures*, page 10, 2003.
- [4] J. Goldstein. Emergence as a construct: History and issues. *Emergence*, 1(1), 1999.
- [5] F. Heyligen. Self-organization, emergence and the architecture of complexity. In *Proceedings of the 1st European Conference on System Science*, Paris, 1989.
- [6] F. Heyligen. The science of self-organisation and adaptivity. In *The Encyclopedia of Life Support Systems*. UNESCO Publishing-Eolss Publishers, Oxford, UK, 2002.
- [7] T. Hogg and B. A. Huberman. Controlling chaos in distributed systems. *IEEE Trans. on Systems, Man and Cybernetics*, 21(6):1325–1332, November/December 1991.
- [8] J. Holland. *Emergence: from Chaos to Order*. Addison-Wesley, 1998.
- [9] B. A. Huberman and T. Hogg. The emergence of computational ecologies. In L. Nadel and D. Stein, editors, *1992 Lectures in Complex Systems*, volume V of *SFI Studies in the Sciences of Complexity*, pages 185–205. Addison-Wesley, Reading, MA, 1993.
- [10] IBM. *Autonomous Computing: IBM's Perspective on the State of Information Technology*. IBM, 2001.
- [11] Y. Indrayadi. *Distributed Dispatching Control For Dynamic Flow-Line Manufacturing Systems*. PhD thesis, Katholieke Universiteit Leuven, September 2002.
- [12] M. Koebe and G. Mayer-Kress. Use of recurrence plots in the analysis of time series data. In M. Casdagli and S. Eubank, editors, *Nonlinear Modelling and Forecasting*, volume XII of *SFI Studies in the Sciences of Complexity*. Addison-Wesley, 1991.
- [13] P. J. Moylan and D. J. Hill. Stability criteria for large-scale systems. *IEEE Transactions on Automatic Control*, AC-23(2):143–149, April 1978.
- [14] U. Nehmzow and K. Walker. Is the behaviour of a mobile robot chaotic? In *Proceedings of the AISB'03 Symposium on Scientific Methods for Analysis of Agent-Environment Interaction*, pages 12–19, 2003.
- [15] J. Odell. Agents and complex systems. *Journal of Object Technology*, 1(2):35–45, July-August 2002.
- [16] J. Odell. Objects and agents compared. *Journal of Object Technology*, 1(1):41–53, May-June 2002.
- [17] H. V. D. Parunak. A dynamical systems perspective on agent-based 'going concerns'. Technical report, Center for Electronic Commerce, 1998.
- [18] H. V. D. Parunak and S. Brueckner. Entropy and self-organization in multi-agent systems. In J. P. Müller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 124–130, Montreal, Canada, 2001. ACM Press.
- [19] H. V. D. Parunak, R. Savit, and R. L. Riolo. Agent-based modeling vs. equation-based modeling: A case study and users' guide. In *MABS*, pages 10–25, 1998.
- [20] H. V. D. Parunak, R. Savit, R. L. Riolo, and S. J. Clark. Dasch: Dynamic analysis of supply chains. Technical report, CEC/ERIM, 1999. Final Report.
- [21] V. Parunak and R. VanderBok. Modeling the extended supply network, 1998.
- [22] H. Van Parunak. Complexity theory in manufacturing engineering: Conceptual roles and research opportunities. Technical report, Industrial Technology Institute, 1993.
- [23] H. Van Parunak. The heartbeat of the factory: Understanding the dynamics of agile manufacturing enterprises. Technical report, Industrial Technology Institute, 1995.
- [24] R. S. VanderBok and H. V. D. Parunak. Managing emergent behavior in distributed control systems, 1997. ISA TECH.
- [25] F. Zambonelli and H. V. D. Parunak. Signs of a revolution in computer science and software engineering. In *Engineering Societies in the Agents World III, Third International Workshop, ESAW 2002, Madrid, Spain, September 16-17, 2002, Revised Papers*, volume 2577 of *Lecture Notes in Computer Science*. Springer, 2003.