5

Complexity and Emergence in Engineering Systems

Chih-Chun Chen[1], Sylvia B. Nagl[2], and Christopher D. Clack[1]

[1] Department of Computer Science, University College London
[2] Department of Oncology and Biochemistry, University College London
c.chen@cs.ucl.ac.uk

Abstract. The purpose of this chapter is to introduce the reader to the key concepts of complexity and emergence, and to give an overview of the state of the art techniques used to study and engineer systems to exhibit particular emergent properties. We include theories both from complex systems engineering and from the physical sciences. Unlike most reviews, which usually focus solely on one of these, we wish to analyse the ways in which they relate to one another, as well as how they differ, since there is often a lack of clarity on this.

5.1 Introduction and Chapter Outline

The world economy, human body, financial markets and world wide web are just a few examples of complex systems with emergent properties. Such systems are composed of a set of constituents that together, through their interactions, give rise to one or more higher level properties or behaviours. These properties are often difficult to predict because interactions between the constituents are non-linear (small differences at the local level can lead to very different outcomes at the global level) and the structure of their interactions tends to change dynamically. Scientists seeking to understand such systems therefore require special analytical methods and tools. At the same time, designers and engineers of information and communication systems have sought to exploit and control emergent properties arising from interactions between system components.

Figure 5.1 shows a map of Complex Systems research. The two major disciplines are Complexity Science and Complex Systems design & engineering.

Complexity Science can be further subdivided into generic concerns, where the goal is to develop special methods and tools for understanding complex systems in general, and domain-specific disciplines, which study a particular real-world complex system using these special methods. There is significant dialogue and exchange between these two branches since the development of methods can be driven by domain-specific needs.

Complex systems design and engineering is concerned with developing systems that exploit emergent properties. These can be situated multi-component

A. Tolk, L.C. Jain (Eds.): Comp. Sys. in Knowledge-based Environments, SCI 168, pp. 99–127.
springerlink.com

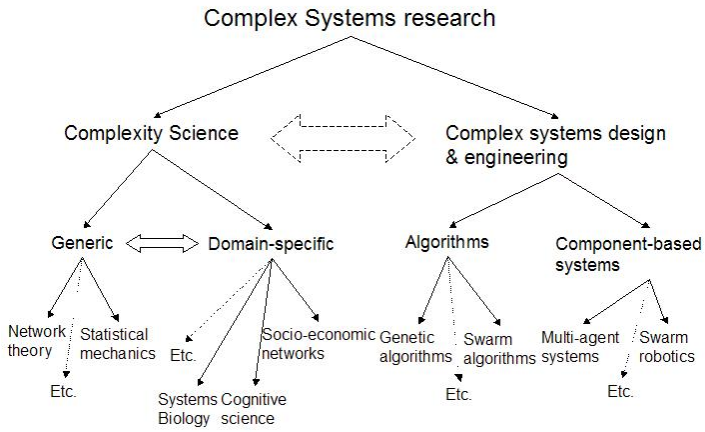

Fig. 5.1. Map of Complex Systems research. There is currently a lack of much-needed dialogue between Complexity Science and Complex systems design & engineering.

systems, which satisfy some function through the collective action of the components, or algorithms, where a solution to a computational problem is arrived at in a distributed fashion. Complex systems design and engineering is also concerned with controlling emergent system behaviours so the system's behaviour remains within an acceptable range.

Although Complexity Science and Complex Systems engineering have many common concerns, dialogue between them is lacking. This chapter introduces the fundamental complex systems concepts and constructs from both the complex systems engineering and complexity science perspective. By considering the work of both these groups within a common framework, we seek to open up the possibility for dialogue by clarifying the main commonalities and differences.

5.1.1 Chapter Outline

The chapter will be structured as follows:

- Section 5.2 introduces the key concepts of complexity, emergence and self-organisation, reviewing the different positions taken in the field.
- Section 5.3 compares the main theories of emergence, focusing especially on those for distributed and multi-agent systems.
- Section 5.4 will critically review techniques that have been used to specify and engineer specific emergent behaviours.
- Section 5.5 concludes the chapter and outlines important challenges remaining in the field.
- Section 5.6 suggests readings and resources for readers wishing to further pursue the topics described in this chapter.

5.1.2 Introductory Concepts and Terminology

Since this chapter assumes familiarity with certain terminology, and because terms are often used ambiguously, we first define the main terms used in the chapter.

Systems, Subsystems and System Components. A system is a set of units, called its components or constituents, that are related to one another in some way. Any subset of the system's components is a subsystem. Since much of the work in this area is in relation to multi-agent systems and simulations, we will also refer to agents, which are the components of multi-agent systems.

Properties, Behaviours and Functions. A property of a component or system is anything about the component or system that can be detected. For example, a component in a particular state X can be said to have the property of being in state X i.e. state X is a property of the component. On this definition, structures, patterns and behaviours are all properties.

When a system is in operation, the states of its components are changing through time. The collective effect of these changes in state during the course of the system's lifetime can then be said to be its behaviour. Similarly, the behaviour of a component can be defined as the sequence of state changes it undergoes during a specified period of time.

A designed system exists to serve some purpose, which is defined by the stakeholders and users of the system. The properties that the system must possess to realise this purpose are its functions. Similarly, the functions of a system component are the properties it must possess to contribute to the system realising its purpose.

Relationships and Interactions. Components in a system can be related to each other in various ways. For example, one components might *contain* another or be *coupled with* another. A relationship can be said to exist between two or more components when the state of one component is in some way related to the state(s) of another component(s). The same applies to systems and subsystems.

Interactions are a special type of relationship. To say that one component interacts with another is to say that its state plays some causal role in determining the state of another component. For example, in a rule-based system where rules are fired when a condition is fulfilled, an interaction between components A and B can be defined as the case where the firing of one of A's rules is the consequence (either full or partial) of B being in a particular state. In contrast, non-interactive relationships are non-causal and can usually be reduced to a question of definition. For example, in a nested system where a component A contains another component B, we might define states such that A is in state q_{A1} when B is in q_{B1}.

5.2 Complexity, Emergence and Self-organisation

Complexity, emergence and self-organisation are terms that are often found together. The relationship between them can be summarised as follows:

> *Emergence* occurs in a system when the system's *complexity* increases by self-organisation i.e. without external intervention.

Given the relationship between emergence and complexity, it is clear that any definition of emergence must assume some measure of complexity. However, to capture the various intuitions as to how the measure should change with particular system features, the measure adopted often depends on the particular domains and/or application.

5.2.1 Complexity Measures

The diversity of complexity measures described below and summarised in Table5.1 reflects the fact the lack of consensus in this area. Because different measures behave differently with repsect to certain system features (e.g. number of components, number of modules), they serve different purposes more effectively and therefore have different applications (also given in Table 5.1). For example, for systems with hierarchical structures, measures relating to modularity may be more useful than those that only consider algorithmic or statistical complexity.

It is important to distinguish between the complexity of a system and the complexity of the design of the system. The design of a system is the set of rules or the program that generates it. Algorithmic complexity, statistical complexity and connectivity are measures that apply to the system itself since they try to determine how comlplex the design would have to be to generate the system[1]. On the other hand, design size, logical depth and sophistication directly address the design of the object or the program used to generate it. Structure and organisation can apply to both the design of the system and the system itself.

Measures can also be characterised according to how they behave when presented with different systems. For example, algorithmic and statistical complexity behave similarly for systems exhibiting regularity but differently for systems with random patterns of behaviour. The algorithmic complexity of a completely random system would be maximum (since each randomly generated event would have to be stored), but such a system would have a low statistical complexity since the system would be one of many (with which it would have no statistically significant similarities) that could be generated by the random source [49], [9].

Recently, complexity measures that do not take into account the hierarchical structure of the object/system, such as algorithmic complexity and connectivity

[1] However, for designed systems, the number of symbols in the design program is often used as an approximation for the algorithmic complexity e.g. [54]. This is based on the assumption that the design program is also the minimal program, but it is not possible to show this conclusively i.e. that a shorter program could not have generated the system.

Table 5.1. System and design complexity measures

Complexity measure	Definition	Main application
Alogrithmic Complexity	Number of symbols of the shortest program that produces an object. The value for algorithmic complexity is always an approximation, since it is not possible to determine with certainty what the minimal program would be.) E.g. [17], [61].	Data generated by a program.
Statistical Complexity	Number of symbols of the shortest program that produces the statistically significant features of an object. E.g. [90], [91].	Data (usually time series) generated by a program.
Connectivity	The number of edges that can be removed before the graph representing the object is split into two separate graphs. E.g. [97], [35].	Objects with highly connected interacting components.
System structure and organisation	Function of the degree of connectivity between and within subsets of components. E.g. [97].	Objects with sophisticated hierarchical structures.
Design size	The length in symbols of the assembly procedure for an object. E.g. [54]	
Logical depth	Computational complexity of the assembly procedure for an object i.e. the times it takes to compute the assembly procedure. E.g. [8]	
Sophistication	The number of control symbols in the program that generates the object. E.g. [62].	
Grammar size	The number of production rules in the program required to produce an object. E.g. [35].	
Design structure and organisation	Function of the number of modules, the reuse of these modules and the degree of nesting. E.g. [54].	

(alone), have been criticised for being counter-intuitive [97], [54]. It is argued that the complexity measure should be such that complexity increases both with increasing numbers of modules and with the degree of intergration between

these modules. The results of tests with different types of systems to see how different complexity measures behave are given in [54].

System Complexity. Here, we consider four measures of complexity addressing a system directly: (i) Algorithmic complexity; (ii) Statistical complexity (see Figure 5.2); (iii) Connectivity and (iv) Structure and organisation.

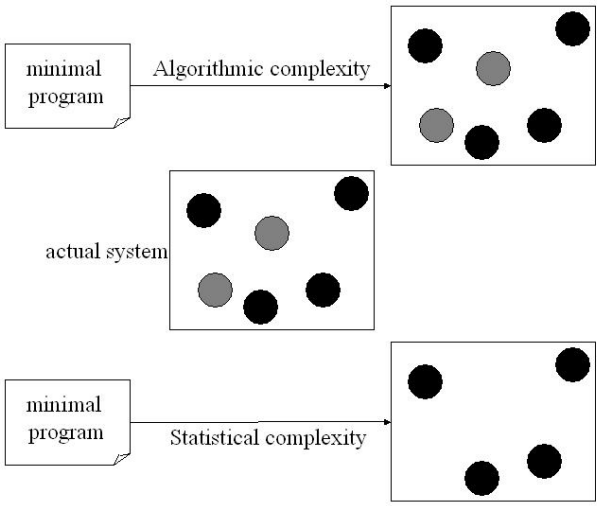

Fig. 5.2. Algorithmic complexity is the length of the minimal program that is able to generate the actual system/object whereas statistical complexity is the length of the minimal program that is able to generate the statistically significant aspects of the system

Algorithmic complexity [17], [61]. The algorithmic complexity of a system is the length of the smallest program that is able to generate the system. The shortest program can also be thought of as the most efficient design for the system. More formally, if we were to characterise the system as a string, the algorithmic complexity would be the number of symbols in the smallest set of grammar rules that could generate that string. (See Figure 5.2)

Statistical complexity [90], [91]. The assumption underlying statistical complexity is that the system is one a set of possible systems that can be generated by a common source. The statistical complexity of the system is the length of the smallest program that is able to generate the statistically significant aspects of the system. These are the aspects that the system shares with the other systems that can be generated by the source.[2] (See Figure 5.2)

[2] Algorithmic and statistical complexity have in common the fact that they try to infer something about the complexity of the generating source or design of the system while connectivity and structure relate directly to the system itself.

Connectivity. The connectivity of a system refers to the extent to which its components are interconnected. It is easy to see this as a network or graph structure, where each of the nodes are the components [35]. Connectivity can then be determined by counting the maximum number of connections that can be removed before the system is split into two separate networks. Connectivity does not have to be static. In fact, one of the key features of complex systems is the fact that connectivity is dynamic, with changes in which components interact with one another i.e. the graph topology changes over time [44]. Sophisticated measures of connectivity are used in Neuroscience, which take into account self-connectedness or re-entry and the fact that connections between components are often dynamic in a system and change with time [97]. (See also Figure 5.3)

Structure and organisation. An especially high degree of connectivity between a subset of components can be interpreted as a module. These subsets can also show a relatively high degree of connectivity with each other, resulting in a hierarchical structure. One measure of system complexity is therefore its modularity and hierarchy, where a system with a high degree of modularity and hierarchy has a higher degree of complexity (e.g. [97]).

Design Complexity. Measures of design complexity are those that relate to the design of the system components and the way they should relate to one another in a system. With traditional Systems design and engineering techniques, the distinction is more difficult to draw, since the system complexity is often a straightforward function the design complexity. However, if a system's properties or behaviours can not be derived from the design because the interactions and connections between the components are dynamic, the relationship may not be as straightforward (see Figure 5.3).

Here, we consider five design complexity measures: (i) Design size; (ii) Logical depth; (iii) Sophistication; (iv) Grammar size and (v) Structure and organisation.

Design size. Design size is the number of symbols in the design program.

Logical depth. Logical depth is the computational complexity of the system's design program. Unlike design size, this would also take into account the number of times a particular procedure is executed to produce the system.

Sophistication. Sophistication is the number of control symbols (e.g. selection statements, loops) in the program that generates the system.

Grammar size. Grammar size is the number of rules in the design program that produces the system.

Structure and organisation. In [54], structure and organisation includes measures of modularity, hierarchy and reuse. Modularity refers to the number of encapsulated group of elements in the design program that can be manipulated as a unit (in [54], this includes procedures). Hierarchy refers to the number of nested

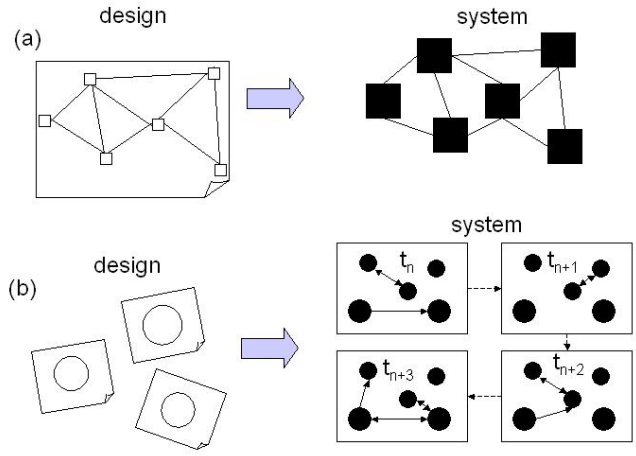

Fig. 5.3. Design and system complexity. (a) In traditional design and engineering paradigms, the system complexity can be established analytically from the design complexity. (b) Paradigms that exploit emergent properties and behaviours are those where the relationship between design complexity and system complexity can not be established analytically simply from the design (even if they are discovered empirically to be related in some way). This is because in the system, the relationships and interactions between the components tend to be dynamic and constantly changing in ways that are not explicitly defined in the design.

layers of modules. Reuse is a measure of the average number of times elements of the design program are used to create the resulting design. Several measures are possible, depending on which elements are counted (e.g. reuse of build symbols vs. reuse of modules; see [54] for more details).

5.2.2 Self-organisation

Self-organisation is the process whereby some system property occurs solely from the behaviours and interactions between the system's components.

Theories of self-organisation tend to fall into one of three categories:

1. Complexity-based theories, which emphasise the description of the process of itself and characterise it as a shift in complexity.
2. Design-oriented theories, which emphasise the discrepency between the design of the system components and the functions the system is able to perform as a whole (without these being explicitly specified in the componenents' deisgns). Self-organisation is the mechanism by which this discrepency is able to exist.
3. Environment-oriented theories, which focus on the ability of the system to adapt to its environment and tend to be interested in the occurrence of different system properties in response to different environments through self-organisation.

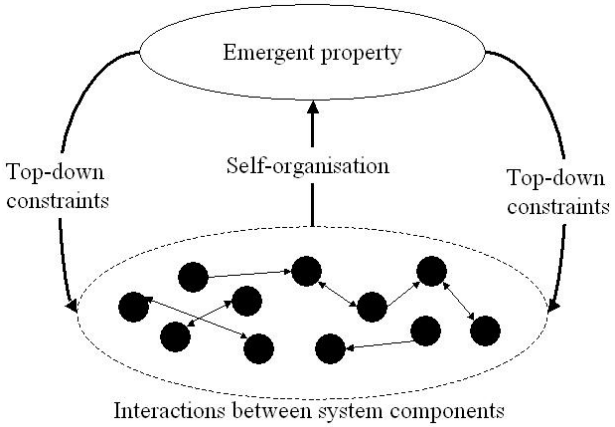

Fig. 5.4. Lower level properties give rise to a higher level emergent property, which in turn constrains the set of lower level properties that can be realised.

If a property or behaviour arising through self-organisation tends to promote itself, we say it is exhibiting **autopeoisis** [99]. Autopeoitic systems are those where the system components, through their interactions with one another and their individual behaviours, regenerate a set of interaction relationships (an interaction network). In the way, the interaction network reinforces and perpetuates itself. This idea is also closely linked with 'downward causation', where an emergent property constrains the properties that can be realised at lower levels; due to these constraints, the set of higher level properties that are able to emerge is also constrained (since these are dependent on the lower level properties). This is illustrated in Figure 5.4.

In the Artificial Intelligence and Artificial Life literature, other 'self-*' terminology [31] is also used (e.g. self-managing, self-optimising, self-protecting). In the current discussion, we subsume these under self-organisation, since they are essentially special categories of self-organisation. However, we also believe that distinguishing between different sub-categories of self-organising properties when studying them is likely to lead to more detailed insight into the mechanisms unique to each category.

Dynamic Shifts in Complexity. On one interpretation, self-organisation is the dynamic process that occurs when a system shifts from one level of complexity to another without any external input or guiding force (see Section 5.2.2). This interpretation relates to the complexity measures for the actual system (see Section 5.2.1 above) rather than to those for its design. The shifting of complexity is called emergence (see Figure 5.5).

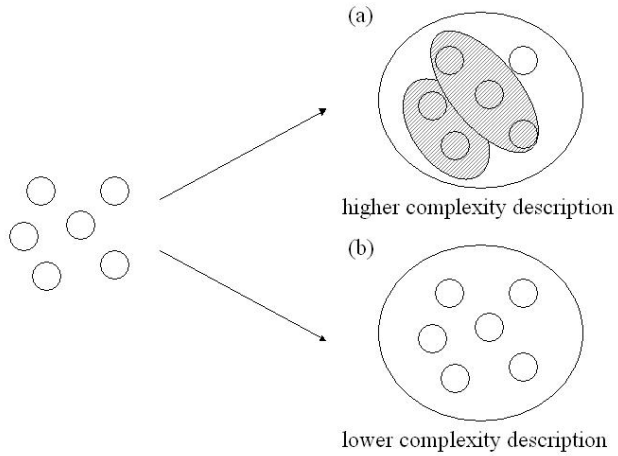

Fig. 5.5. Shifts in complexity. A higher (a) or lower (b) complexity description of a system may emerge as the result of self-organisation.

A shift in complexity means a change in the description of the system. If we consider again the four measures of system complexity introduced above, we can interpret complexity shifts as follows:

Algorithmic Complexity. A decrease in algorithmic complexity indicates that there is another algorithm or design that can generate the system's behaviour more efficiently so that the current algorithm is no longer the shortest while an increase indicates that the current algoroithm is no longer sufficient to generate the system's behaviour.

Statistical Complexity. Similarly, a decrease in statistical complexity indicates that there is a shorter algorithm that can generate the statistically significant aspects of the system's behaviour while an increase indicates that the current algorithm is no longer sufficient to generate the statistically significant features of the system's behaviour.

Connectivity. A change in connectivity means that a different number of components in the system are connected to one another. This implies that a higher or lower number of the components are involved in driving the system forward in its evolution i.e. a different number of components are required for the system to behave in a particular way.

Structure and organisation. A change in structure and organisation is reflected in changes in connectivity between subsets of components. For example, highly connected components that initially functioned as modules might show a

reduction in connectivity so that the modularity and complexity of the system decreases.

Design/Observation Discrepency in Multi-component Systems. With a more design-oriented interpretation, we might also say that a self-organised system is one where the behaviour exhibited by the system is not explicitly given in the design of any of its components. Instead, the interactions between the components together give rise to some property or behaviour of the system with no controlling component. System properties and behaviours that occur in this way (with no top-down control) are said to be emergent and the components of the system are often said to exhibit cooperative or coordinated behaviour (see Section 5.2.2).

In systems which are intended to perform a particular function, self-organisation can be seen as the system's ability to perform this function without this being explictly incorporated in the design. This means there is no one component or set of components whose purpose in the system is to ensure that the other components' behaviours are coordinated in such a way that the function is performed. Instead, the components collectively give rise to an overall system behaviour that performs the function.

Adaptivity to the Environment. Finally, a more environment-oriented and functional view of self-organisation focuses on the ability of the system to adapt to its environment in ways that have not been considered in the design 'without explicit external command..' [87].

5.2.3 Emergence

Fundamental to the concept of emergence is the idea that there is a change in the way system components interact and relate to one another. Depending on the complexity measure chosen to characterise the descriptions of the system and/or its design, emergence can be seen to result in an increase or decrease in complexity. For example, in [10], emergence is defined as an unexpected complexity drop in the description of the system by a certain observer. When an observer observes a system at work, he has a theory about the rules governing the system. If the observer understands very little about the system, the set of rules he associates with the system's behaviour will be very large, since every component of the system will have to be described independently. When the observer understands more about the system, he might be able to characterise its behaviour using a smaller set of rules. This smaller set of rules represents the lower complexity description and can be said to emerge from the larger set of rules representing the higher complexity description.

5.2.4 Summary and Analysis

The relationship between the different theories of complexity, self-organisation and emergence can be described by the following logical chain:

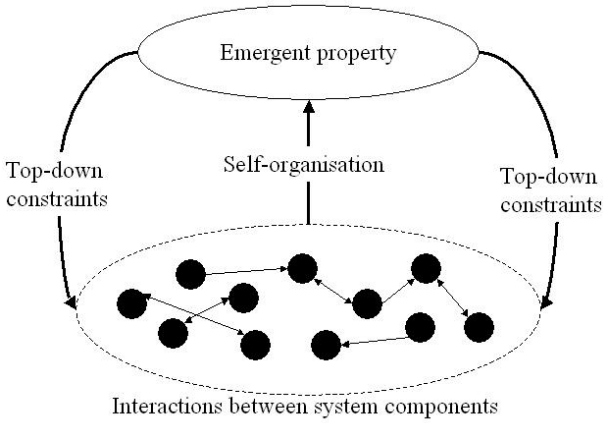

Fig. 5.6. Lower level properties give rise to an emergent property, which in turn constrains the set of lower level properties that can be realised.

> Interactions between components → Self-organisation → Complexity → Emergent property → Constraints on interactions between components

An important point to note is that the chain is not supposed to represent a temporal or causal sequence. Interactions between components both give rise to and constitute the emergent property, which in turn constrains the interactions between the components. The phenomenon of autopeoisis results when the constraints on the interactions perpetuate those same interactions. This is illustrated in Figure 5.6.

Systems usually operate in a dynamic (as opposed to static) environment so that interactions between the system and the environment can influence the system's operation (see Figure 5.7), giving the following logical chain (adapted from [22]):

> Environmental Change → Change in interaction graph[3] → Shift in complexity → Self-organisation → Emergence of new property

Again, no member of the chain is temporally or causally prior to any of the others.

Unlike traditional design and engineering paradigms, designs that exploit emergent properties and behaviours are those where the resulting system has features that are not explicitly specified in the design. For this reason, their

[3] An interaction graph represents the interaction relationships between a system's components. Each node in the graph represents a component and the edges represent the interactions.

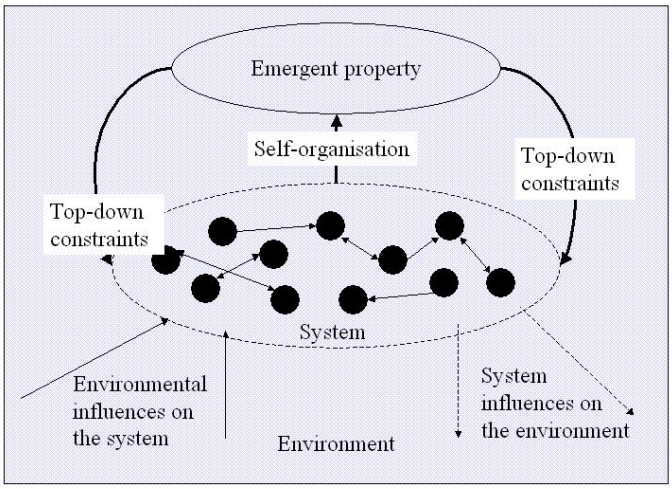

Fig. 5.7. The environment and the system together give rise to an emergent property, which in turn constrains the properties and interactions between the system constituents. If the system is also able to influence its environment, then the emergent property would also constrain system-environment interactions via the constraints on the system itself.

complexity can not be established analytically from the design alone (see Figure-fig:TraditionalNew). More formally, if C_{design} stands for the design complexity, C_{system} stands for the system complexity, and $C_{system} = f(C_{design})$, for traditional design and engineering paradigms, we can establish what the function f is from the design whereas for paradigms exploiting emergence, we can not.

5.3 Theories of Emergence

Emergence has been a much-discussed subject in both the physical sciences and in engineering. Unfortunately, it is often seen as an area where there is little consensus and much confusion. However, we shall try to show that many of the apparent differences between definitions are not fundamental but simply due to different aspects of the phenomenon being emphasised.

5.3.1 Formalising the Micro-macro Relationship

The idea that a system can be observed and described at different levels of abstraction is central to most definitions of emergence e.g. [23], [25], [10], [26], [83], [86], [89], [5], [63].

In the Design and Engineering domain, this has tended to be formalised uing language-oriented definitions based on multi-agent systems (e.g. [26], [83], [63]), while those inspired by statistical mechanics used to study real systems (both

physical and social) tend to focus on relating the different scopes and resolutions of properties (e.g. [23], [89], [55], [56], [85]).

Language-oriented Definitions. Language-oriented definitions (e.g. [26], [63]) require that the macro-level description is in some way 'greater than' the micro-level description i.e. the macro-level language has elements that can not be found in the micro-level language.

A grammar can be seen as a formal computational device with a particular generative power i.e. it is able to generate a particular language or set of languages. In Kubik's formalisation of emergence [63], the micro-level grammar L_{PARTS} is determined by the sum of conditions that agents can bring about in the environment if they act individually in the environment. If the multi-agent system can generate a language L_{PARTS} that can not be generated by the summation of individual agents' languages, it is said to have an emergent property.

Similarly, Demazeau's definition [26] refers to the discrepancy between the language represented by the whole system and the langauge represented by summing the parts. A description of the whole system consists of agents (A) in their environment (E) using interactions (I) to form an organisation (O). However, if we simply summed these different elements ($A + E + I + O$), we would not get the same result as if we considered the system as a whole.

Since the micro-macro discrepancy on these definitions is inextricably linked to the design/observation discrepancy on these definitions, there is no reference to shifts in system complexity.

Hierarchies, Scope and Resolution. In [55] and [56], two categories of hierarchy are described (see Figure 5.8):

1. Compositional hierarchy, where lower level properties are constituents of higher level properties. This can be seen to correspond to α-aggregation, the AND relationship, or part-whole;
2. Specificity or type hierarchy where higher level properties are defined at a lower resolution than lower level properties. This can be seen to correspond to β-aggregation [55, 56], or the OR relationship.

We can relate these two categories of hierarchy to the account of micro-macro-property relationships given in [85], which defines a property P_1 to be a macro-property of another property P_2 if:

- P_2 has a greater scope than P_1;
- P_2 has a lower resolution than P_1; or
- both.

The scope of a property is the set of constituents required for the property to exist; for example, the property of being a flock requires a minimum number of birds. On the other hand, the resolution is the set of distinctions that have to be made to distinguish the property; for example, to identify a colour, one

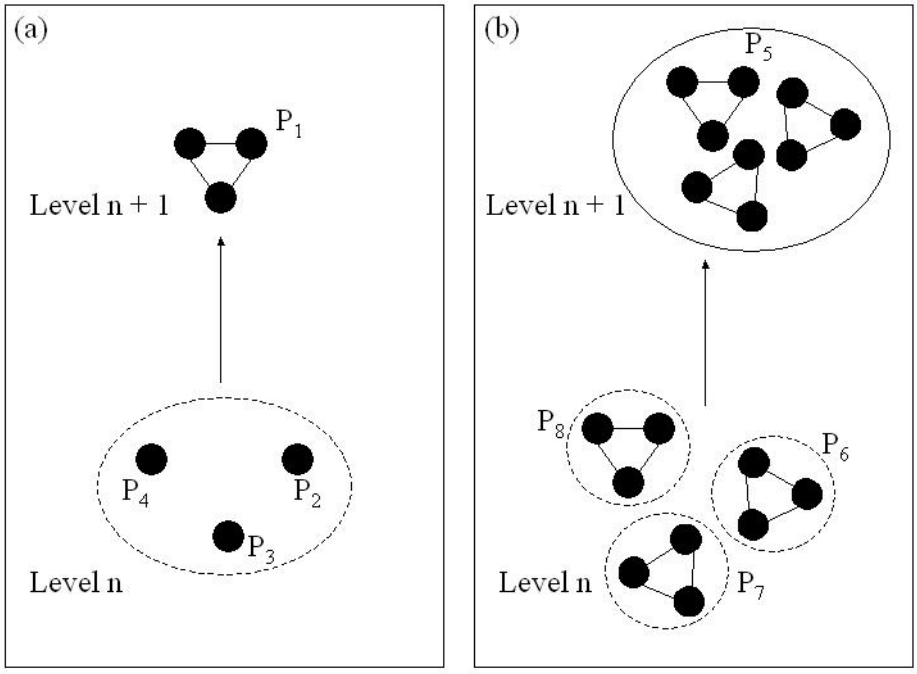

Fig. 5.8. Two categories of hierarchy. (a) Compositional hierarchy/α-aggregation: P_2, P_3 and P_4 are constituents of P_1. (b) Type hierarchy/β-aggregation: P_6, P_7 and P_8 fall in the set defined by P_5.

needs to be able to distinguish between a ranges of wavelengths. Information-theoretic interpretations of emergence in dynamic systems are based on the idea that often, when we are considering a greater scope, we are willing to accept some loss of accuracy or a lower resolution when predicting future behaviour (see, for example [10], [88]).

5.3.2 Formalising the Design-Observation Discrepancy: Definitions of Emergence for Designed Multi-agent Systems

In the context of designed multi-agent systems, a defining characteristic of emergent properties and behaviours is that they arise 'spontaneously' without being explicitly specifed in the design. In other words, it is not possible to predict their occurrence simply from looking at the design program. For example, Ronald et. al. [83] suggest that a property can be said to be emergent if (i) the system has been constructed from a design describing the interactions between components in a language L_1, (ii) the observer is fully aware of the design but describes the behaviour of the system using langauge L_2, (iii) L_1 and L_2 are disinct and (iv) The causal link between the interactions described in L_1 and the system behaviour described in L_2 is non-obvious. This is somewhat controversial since it seems to make the

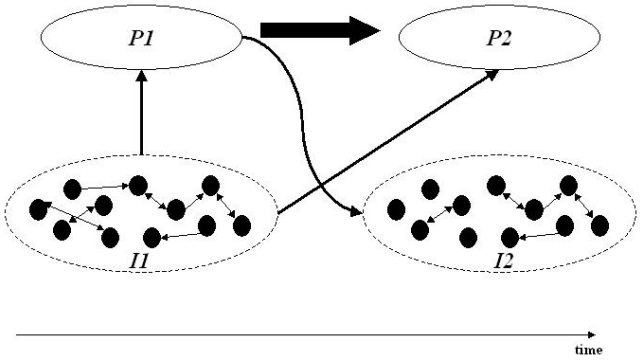

Fig. 5.9. A 'law' emerges when an emergent property $P1$ constrains the properties and/or interactions of its constituents $I1$ so they give rise to another emergent property $P2$ made up of interacting constituents $I2$

emergence classification of a property dependent on the observer's knowledge i.e. whether or not the observer thinks the causal link between the L_1 property and L_2 property is non-obvious.

A more objective criterion is given by Darley [25], who defines an emergent property as one 'for which the optimal means of prediction is simulation'. In other words, given the design, it can only be deduced by stepping the evolution of the system, that the property will be present.

5.3.3 Top-Down 'Causation' and Emergent 'Laws'

An important feature of emergent properties is that they can constrain or influence the properties of their components. Some even hold the position that this feature is mandatory for a property to be called 'emergent' (e.g. [92], [86]). The phenomenon of 'autopeoisis' described in Sections 5.2.2 and 5.2.4 is an example of this, since an emergent property sustains itself by constraining the interactions between system components so they perpetuate themselves (see Figure 5.10). More generally though, top-down constraints mean that a particular property $P1$ that emerges from a set of component properties and/or interactions $I1$ make the appearance of a second set of interactions $I2$ more likely or certain (e.g. [6]). If $I2$ is also associated with an emergent property $P2$, then a higher level 'law' will emerge that relates $P1$ and $P2$ (this might be deterministic or probabilistic). This is illustrated in Figure 5.9. Computational statistics techniques grounded in information-theoretic interpretations of emergence (see Section 5.3.1) can be used to identify statistical regularities or 'laws' that emerge e.g. [88], [89], [24], [91].

5.3.4 Summary and Analysis

Although there are several definitions of emergence in the literature, definitions tend to address one or more of the following:

1. the micro/macro relationship;
2. the design/observation discrepancy;
3. top-down 'causation' effects.

In the context of designing and engineering systems with emergent properties, these characteristics present us with particular challenges. In particular, the design/observation discrepancy means that it is difficult to specify components to guarantee that their interactions give rise to the desired system behaviour.

5.4 Designing and Engineering Emergent Behaviours in Complex Decentralised Systems

Recently, there has been significant interest in designing systems in such a way that the system's successful operation depends on properties and behaviours that are not explicitly specified in the design i.e. the systems are decentralised and self-organising. For example, multi-agent systems are used in distributed planning and reasoning (e.g. [11], [46], [2]), and swarm algorithms (e.g. [60], [32], [67]) and other nature-inspired techniques such as genetic algorithms [48] and artificial neural networks [66], [64], [76] have been used to solve problems, including those that fall into the NP-complete class [51]. These systems are believed to be more robust and adaptive (see also Section 5.2.2) to their environment [52], [53] than traditionally designed systems with centralised control. In the case of problem-solving algorithms, distributed knowledge and information can be exploited to arrive at a solution.

To exploit the emergent properties of a system, traditional methods for designing, engineering and analysing systems need to be replaced with new paradigms that take into account the discrepency between the design of the individual components and their collective behaviour in a system. Whereas traditional

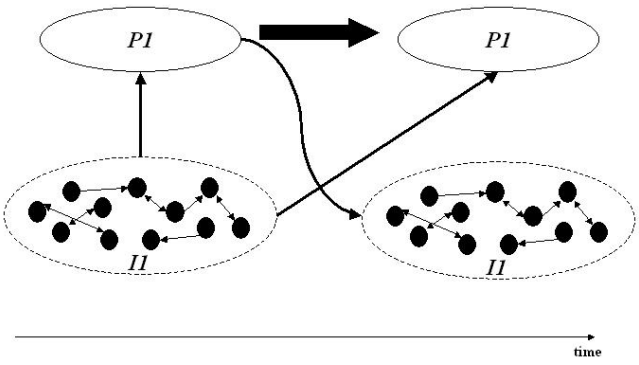

Fig. 5.10. Autopeoisis. An emergent property $P1$ constrains the properties and interactions between its constituents $I1$ so that $I1$ (and hence $P1$ itself) is sustained or perpetuated.

engineering aims to ensure that components' interactions with one another are predictable and controlled for successful system operation, emergence-based engineeering seeks to exploit the dynamic aspect of components' interactions. When a traditionally engineered system fails to deliver its functionality, it is either because one or more of the components has failed or because a control has failed to operate. When an emergence-exploiting system fails to deliver its functionality however, it is usually because the components have interacted in such a way as to give some other, undesired, emergent behaviour.

Importantly, the difference between traditional methods and emergence-based methods lies in the different *relationship* between the system's *design* and the system's *functionality*, not necessarily in any intrinsic feature(s) of the systems themselves. Emergence-based methods exploit the emergent features of the system and design the system in such a way as to promote the appearance of these emergent features whereas traditional methods do not exploit the emergent features (even though these may still appear[4]). This is expressed succinctly in [15], [43] and [45] by the following conditions that emergent-based designs satisfy:

1. The goal of a computational system is to realise an adequate function, judged by a relevant user. This function might be a behaviour, a pattern, or some other property that has to emerge.
2. 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. 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 functin nor how to reach it.

The methods and techniques reviewed in this section come mainly from the field of multi-agent systems. As with traditional Systems Engineering, the process of designing and then implementing the system is usually an iterative one that cycles (often several times) through the phases of:

1. Requirements gathering and analysis;
2. Design;
3. Construction (in software systems, this would include coding);
4. Testing.

By definition, if we are exploiting the emergent properties of a system, it is not possible to predict with certainty that we will be able to from the design of its components alone. However, it may be possible to establish with a particular margin of error, how likely or with what frequency the property/behaviour will emerge, given a particular design. Choosing an appropriate and efficient method for doing this is itself difficult and can depend on the application.

Gleizes et. al. [45] cite three main challenges in relation to engineering systems so they have the appropriate emergent behaviours for desired functionalities:

[4] In fact, in come cases, the unexpected appearance of these features contributes to system malfunction and is a sign of bad design.

1. Controlling system behaviour at the macro level by focusing on the design of agents at the micro level.
2. Providing designers and engineers with the tools, models and guides to develop such systems.
3. Validating these systems.

Correspondingly, the methods and techniques used in emergent systems design and engineering can be seen to fall into one or more of the following categories:

1. Design methods and methodology that allow designers and engineers to specify a system that exploits emergent behaviours.
2. Validation techniques.
3. Techniques for controlling interactions so that the appearance of desired emergent properties and behaviours is facilitated, and occurrence of undesirable behaviours is prevented.

The first two of these are considered in Section 5.4.1, while the latter is considered in Section 5.4.2.

5.4.1 Design and Validation Methods

Design methods and validation techniques tend to be closely related. In the design phase, specifications for the system's components are drawn up based on an understanding of how these components will interact when the system is in operation. These specifications themselves have to be validated to ensure they correctly reflect the requirements. Formal methods provide the strongest form of validation for this purpose (Section 5.4.1). However, the big challenge in engineering systems with emergent properties is in validating system behaviour as it would be in operation. For this purpose, empirical techniques are required (Section 5.4.1)

Formal Methods to Validate Design Specifications. Formal reasoning frameworks can be used to validate design specifications by proving that the specifications entail the occurence or non-occurrence of certain interactions between components. Strong statements can therefore be made about the way the system should behave when implemented.

Formal methods can be seen to fall into the following categories (although some methods cut across categories) [84]:

1. Model-oriented approaches, which involve the derivation of an *explicit* model of the system's desired behaviour in terms abstract mathematical objects amalgamating the behaviour of the components. These can be further subclassified as:
   - Process algebras e.g. Calculus of communicating systems (CCS) [71], π-calculus [72], [36], ambient calculus [16]. These can be used to describe the interactions between components and prove that certain conditions are satisfied with respect to them.

- Concurrency automata e.g. Petri nets [75], Statecharts [50], X-machines [57], [58], which include the details of components' operational behaviour (X-machines have the additional feature of representing the internal states of components [57]).
- Set-theoretic methods e.g. abstract state machines (ASM) [13], the B-Method [65], Z notation [93].

2. Property-oriented approaches, which allow minimal constraints to be specified. These tend to be algebraic methods, which use axiomatic semantics based on multi-sorted algebras and relate system properties to equations over the system components e.g. BDI logic properties can be mapped down to linear temporal logic (LTL) [12] or branching time logic (CTL) [7].

Most of these frameworks also come have tools for model-checking [20].

Empirical Methods for Validating and Developing Decentralised Systems. Formal methods are unable to address the problem of implementing or (in the case of software systems) programming the components of systems to guarantee that the system will satisfy the design specifications [34], [33]. (This follows from Godel's Incompleteness Theorem [47] and the fact that the 'halting problem' is an undecidable [98].) Although this is also true of non-emergence-based design, designers seeking to exploit emergent properties have the additional challenge of understanding how non-obvious behaviours can arise from a multitude of interactions. Empirical methods are therefore required to validate the system's behaviour. The premise on which these are based is that "The behaviour of a system is only a hypothesis about the system's behaviour - it must be checked by experimentation" [34].

One approach is global validation, which establishes that the system as a whole performs the desired function. Simulation and numerical analysis of the results can also be used to check whether the system's performance falls within an acceptable range. Often, agent-based models and simulations have been used for this purpose.

A related approach is to study certain coordination mechanisms such as those described in Section 5.4.2 and reuse them as design patterns [28], [27], [68], [42]. Analysis of these mechanisms using formalisms such as causal loop diagrams [81] can help us to understand the causal structures underlying them (e.g. positive and negative feedback loops) [96]. This more analytical understanding is important as it allows us to reliably synthesise different coordination mechanisms so they interact appropriately [94]. A formalism for relating behavioural motifs at different levels of abstraction has been introduced in [18] and its application in multi-level hypothesis testing is described in [19].

5.4.2 Regulating Interactions

A significant amount of work has been done to try and understand which component interactions tend to lead to desirable emergent properties and behaviours (and which do not). Once this has been established (either through rigorous

proof or empirical studies; see Section 5.4.1), it is the task of the designer to put in place certain mechanisms to ensure these interactions take place (or, in the case of undesirable properties, do not take place).

Mechanisms for regulating interactions between components in a decentralised distributed system fall into two main categories:

- Protocols and rules, which act as a means of ensuring that the behaviour of every component in the system results in interactions that satisfy a set of constraints. These might require specific agent architectures and/or capabilities; and
- Environmental artifacts and architectures, which allow components to communicate with one another through shared data spaces so that their interactions satisfy a set of constraints constraints.

Here, there is only space to briefly refer to the more commonly cited examples, but the interested reader is advised to follow up on the references. For more details on coordination protocols, the reader should refer to the 'Coordination, Organization, Institutions and Norms in Agent Systems (COIN)' workshop series (http://www.pcs.usp.br/∼coin/). For more details on environmental artifacts and architectures, the 'Environments for Multiagent Systems (E4MAS)' workshop series (http://www.cs.kuleuven.be/∼distrinet/events/e4mas/) is a good place to start.

Coordination Protocols and Local Regulation. Coordination protocols are rules that determine the set of permissible interactions between agents. While the precise set of protocols adopted for a particular system tends to be appli cation-specific, various models of interaction inspired by natural and social systems are often involved. Also, environmental architectures and artifacts (see below) often require specific coordination protocols and/or agent architectures to work e.g. the Influence-Reaction model [40] requires both that agents have a 'physical' aspect and that the environment is active.

Examples of coordination protocols include:

- Cooperation [38] and the detection of cooperation failure [77], where agents can coordinate their actions and share resources to achieve a goal;
- Trust [79], [74] and reputation [80], where agents can evaluate the trustworthiness of their peers and find select suitable partners to interact with to achieve their goals;
- Organisational metaphors such as roles and groups [39], [14], [21], [70], where roles provide context constraints for agents' behaviours;
- Norms obligations and institutions [4], [29], [100] where certain constraints govern agents' interactions with one another in particular situations;
- Gossip, where agents select peers to receive information from [59], [1].

Taxonomies of such protocols can be found in [30], [95] and [68].

Environmental Artifacts and Architectures. Environmental artifacts and architectures are used to mediate agent interactions, providing a means for agents to interact with each other reliably [78]. Examples include:

- Interaction channels, where agents share and filter data by publishing and subscribing to repositories, which is responsible for sending (and blocking) messages to the appropriate agents. Examples include multicast interactions [3], shared memory and tuple-based approaches [41], and event-based interaction [37].
- Sychncronisation mechanisms, which can be centralised or decentralised. These hold the resulting effects of simultaneous actions until it is safe to execute them in the system. In centralised synchronisation (e.g. the Influence-Reaction model [40]), the data structures are accessible to all agents in the system whereas in decentralised synchronisation, the system is split into regions which each hold a group of agents that can act simulateously and independently from other agents in the system [101].
- Overlay networks, which restrict the set of agents that can interact with one another. Usually there are protocols that allow new agents to join and leave a network. Examples include distributed hash tables [82] and 'ObjectPlaces', which has *view* and *role* abstractions [102].
- Stigmergic mechanisms, such as pheremones [73] and fields [69], which allow agents to interact and share data indirectly. As agents move around in their enviornment, they can store data in their current location and this can be accesed by other agents (sometimes only within a given time frame). With fields, the data can themselves propogate according to certain rules that prescribe how they spread.

5.4.3 Summary and Analysis

Designing and engineering complex systems with emergent properties is challenging because by definition, the appearance of these properties can not be established analytically from the design of system's components. However, formal and empirical methods have been proposed to better inform designs:

- Formal methods allow designers of these systems to express their design specifications unambiguously and prove that certain conditions and properties hold for specifications.
- Empirical methods allow certain interaction patterns to be identified and reused in the form of protocols and environmental artifacts.

Perhaps the most important criticism of emergence-based design principles is that although they are "compatible with the good average-case performance... they often conflict with a design's predictability". On the other hand, such systems may provide the only means of solving computational problems that would otherwise be too time-consuming to solve. Furthermore, by applying empirical methods and studying such systems, it is possible that we are able to achieve sufficient predictability (and reliability) for a particular purpose.

5.5 Conclusions

This chapter has introduced the main concepts of complexity, self-organisation and emergence. As well as theories from the design and engineering of complex systems, we have also reviewed theories from the physical and complex systems sciences. Complex systems, whether designed or natural, are difficult to analyse because system properties and behaviours are driven by the collective properties and/or relationships between the system's components. Designing and engineering such systems to exploit their emergent properties and behaviours is therefore inherently difficult; formal specification techniques need to be supplemented with empirical methods and analytical techniques.

At the same time, scientists studying complex systems, either in the abstract or in specific domains such as Finance, Systems Biology, Earth Science, Ecology and Economics, face similar challenges. There therefore needs to be more mutual engagement between complexity scientists and complex systems engineers so that methods and techiques can be shared. One of the reasons such engagement has been slow to come about is the confusion surrounding terminology. This chapter has taken the first step to addressing this by clarifying the main distinctions and commonalities between different theories of emergence and complexity.

5.6 Resources and Suggested Readings

Complexity Science is the umbrella term used to describe disciplines that study complex systems with emergent properties. It includes the development of methods for analysing and modelling such systems (e.g. network theory, dynamical systems analysis, statistical mechanics) as well as more domain-specific concerns (e.g. why do stock markets crash?). The reader wishing to learn more about these should refer to the Bibliography and Resource List.

5.6.1 Bibliography

- Axelrod, R., Cohen, M. D. (2001) Harnessing complexity: Organisational implications of a scientific frontier. Basic Books.
- Bar-Yam, Y (1999) The Dynamics of Complex Systems (Studies in nonlinearity). Perseus Books.
- Flake, G. W. (2000) The computational beauty of nature: Computer explorations of fractals, chaos, complex systems and adaptation. MIT Press.
- Holland, J. (2000) Emergence: From chaos to order. Oxford University Press.
- Resnick, M. (1997) Turtles, termites and traffic jams: Explorations in massively parallel microworlds. MIT Press.
- Waldorp, M. (1992) Complexity: The emerging science at the edge of order and chaos. Simon and Schuster.

Acknowledgements

The work presented in this paper is the result of a multi-disciplinary collaboration between the authors: Chih-Chun Chen (Computer Science: complexity science and multi-agent systems), Sylvia Nagl (Oncology: complexity science, systems biology and philosophy of science), and Christopher Clack (Computer Science: type systems, logics, rule-based and adaptive systems).

References

1. Allavena, A., Demers, A., Hopcroft, J.: Correctness of gossip-based membership protocol. In: Proceedings of the 24th ACM Symposium on the Principle of Distributed Computing (2005)
2. Atkins, E.M., Abdelzhar, T.F., Shin, K.G., Durfee, E.H.: Planning and resource allocation for hard real-time, fault-tolerant plan execution. Autonomous Agents and Multi-Agent Systems Journal (Best of Agents 1999 special issue) (1–2), 57–78 (March/June 1999)
3. Balbo, F., Pinson, S.: Toward a multi-agent modelling approach for urban public transportation systems. In: Engineering societies in the agents world II. Springer, Heidelberg (2001)
4. Barbuceanu, M., Gray, T., Mankovski, S.: Coordinating with obligations. In: Proceedings of the second international conference on autonomous agents, pp. 62–69 (1998)
5. Bedau, M.A.: Downward causation and the autonomy of weak emergence. Principia 3, 5–50 (2003)
6. Beer, R.D.: Autopoiesis and cognition in the game of life. Artificial Life 10, 309–326 (2004)
7. Benerecetti, M., Cimatti, A.: Symbolic model checking for multi-agent systems. In: Proceedings of the model checking and artificial intelligence workshop (MoChArt 2002), held with 15th ECAI, Lyon, France, pp. 1–8 (July 21–26, 2002)
8. Bennett, C.H.: On the nature and origin of complexity in discrete, homogenou, locally-ineracting systems. Found. Phys. 16, 585–592 (1986)
9. Boffetta, G., Cencini, M., Falcioni, M., Vulpiani, A.: Predictability: a way to characterise complexity. Physics Reports 356, 367–474 (2002)
10. Bonabeau, E., Dessalles, J.L.: Detection and emergence. Intellectica 2(25), 85–94 (1997)
11. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, Oxford (1999)
12. Bordini, R., Fisher, M., Visser, W., Wooldridge, M.: Verifying multi-agent programs by model-checking. Autonomous agents and multi-agent systems 12, 239–256 (2006)
13. Borger, E., Stark, R.: Abatrsct State Machines: A method for high-level system design and analysis. Springer, Heidelberg (2003)
14. Cannata, N., Corradini, F., Merelli, E., Omicini, A., Ricci, A.: An agent-oriented conceptual framework for systems biology. Trans. On Comput. Syst. Biol. 3, 105–122 (2005)
15. Capera, D., George, J.P., Glize, M.P.: The amas theory for complex problem solving based on self-organising cooperative agents. In: The First International TAPOCS Workshop at IEEE 12th WETICE, pp. 383–388 (2003)

16. Cardelli, L., Gordon, A.D.: Mobile ambients. In: Foundations of Software Science and Computation Structures: First Interational Conference FOSSACS 1998. Springer, Berlin (1998)
17. Chaitin, G.J.: On the length of programs for computing finite binary sequences. J. Assoc. Comput. Mach. 13, 547–569 (1966)
18. Chen, C.-C., Nagl, S.B., Clack, C.D.: A calculus for multi-level emergent behaviours in component-based systems and simulations. In: Aziz-Alaoui, M.A., Bertelle, C., Cosaftis, M., Duchamp, G.H. (eds.) Proceedings of the satellite conference on Emergent Properties in Artificial and Natural Systems (EPNACS) (October 2007)
19. Chen, C.-C., Nagl, S.B., Clack, C.D.: A method for validating and discovering associations between multi-level emergent behaviours in agent-based simulations. In: Nguyen, N.T., Jo, G.S., Howlett, R.J., Jain, L.C. (eds.) KES-AMSTA 2008. LNCS (LNAI), vol. 4953. Springer, Heidelberg (2008)
20. Clarke, E.M., Grumberg, E.M., Peled, D.A.: Model Checking. MIT Press, Cambridge (2000)
21. Corradini, F., Merelli, E., Vita, M.: A multi-agent system for modelling carbohydrate oxidation in cell. In: Computational Science and Its Applications (ICCSA 2005: International Conference, Part II), Singapore, May 9-12, 2005, Proceedings, pp. 1264–1273 (May 2005)
22. Cotsaftis, M.: In: Aziz-Alaoui, M.A., Bertelle, C., Cotsaftis, M., Duchamp, G.H.E. (eds.) Proceedings of EPNACS 2007, Emergent Properties in Natural and Artificial Systems, Dresden, Germany, October 1–5, pp. 9–33 (2007)
23. Crutchfield, J.P.: The calculi of emergence: Computation, dynamics, and induction. Physica D 75, 11–54 (1994)
24. Crutchfield, J.P., Feldman, D.P.: Regularities unseen, randomness observed: Levels of entropy convergence. Chaos 13(1), 25–54 (2003)
25. Darley, V.: Emergent phenomena and complexity. Arificial Life 4, 411–416 (1994)
26. Demazeau, Y.: Steps towards multi-agent oriented programming. In: First International Workshop on Multi Agent Systems, Boston, Mass. (1997)
27. DeWolf, T., Holvoet, T.: A catalogue of decentralised coordination mechanisms for designing self-organising emergent applications. Technical Report CW 458, Department of Computer Science, K. U. Leuven (2006)
28. DeWolf, T., Holvoet, T.: Decentralised coordination mechanisms as design patterns for self-organising emergent applications. In: Proceedings of the Fourth International Workshop on Engineering Self-Organising Applications, pp. 40–61 (2006)
29. Dignum, F., Morley, D., Sonenberg, L., Cavedon, L.: Towards socially sophisticated bdi agents. In: Proceedings of ICMAS 2000 (2000)
30. d'Inverno, M., Luck, M.: Understanding agent systems, ch. 3, pp. 39–66. Springer, Heidelberg (2001)
31. Dowling, J., Cunningham, R., Curran, E., Cahill, V.: Component and system-wide self-* properties in decentralized distributed systems. In: Self-Star: Internatinal Workshop on Self*- Properties in Complex Information Systems (2004)
32. Eberhart, R.C., Shi, Y.: Particle swarm optimization: developments, applications and resources. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 27–30 (2001)
33. 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)

34. Edmonds, B., Bryson, J.: The insufficiency of formal design methods - the necessity of an experimental approach - for the understanding and control of complex multi-agent systems. In: Proceedings of AAMAS, pp. 938–945 (2004)
35. Edmunds, B.: Syntactic measures of complexity. PhD thesis, University of Manchester (1999)
36. Esterline, A., Rorie, T.: Using the π-calculus to model multi-agent systems. In: Greenbelt, M.D. (ed.) Proceedings of the First International Workshop on Formal APproaches to Agent-Based Systems, vol. 1871. Springer, Heidelberg (2001)
37. Eugster, P., Felber, P., Guerraoui, P., Kermarrec, A.: The many faces of publish/subscribe. ACM Computing Surveys 35(2), 114–131 (2003)
38. Ferber, J.: Multi-Agents Systems: An Introduction to Distributed Artificial Intelligence. Addison-Wesley, Reading (1999)
39. Ferber, J., Gutknecht, O.: A meta-model for the analysis and design of organisations in multi-agent systems. In: Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS 1998), pp. 128–135. IEEE Computer Society Press, Los Alamitos (1998)
40. Ferber, J., Muller, J.-P.: Influences and reaction: A model of situated multiagent systems. In: Second international conference on multi-agent systems, AAAI (1996)
41. Freeman, E., Hupfer, S., Arnold, K.: JavaSpaces principles, patterns, and practice. Addison-Wesley, Reading (1999)
42. Gardelli, L., Viroli, M., Omicini, A.: Design patterns for self-organising multiagent systems. In: Proceedings of EEDAS 2007 (2007)
43. George, J.P., Gleizes, M.P.: Experiments in emergent programming using self-organising multi-agent systems. In: Pěchouček, M., Petta, P., Varga, L.Z. (eds.) CEEMAS 2005. LNCS (LNAI), vol. 3690, pp. 450–459. Springer, Heidelberg (2005)
44. Giavitto, J.-L., Michel, O.: Mgs - a rule-based programming language for complex objects and collections. Electronic Notes in Theoretical Computer Science, 59 (2001)
45. Gleizes, M.-P., Camps, V., George, J.-P., Capera, D.: Engineering systems which generate emergent functionalities. In: Engineering Environment-Mediated Multiagent Systems (EEMMAS 2007). LNCS. Springer, Heidelberg (2007)
46. Gmytrasiewicz, P.L., Durfee, E.H.: Rational coordination in multi-agent systems. Autonomous Agents and Multi-Agent Systems Journal 3(4), 319–350 (2000)
47. Godel, K.: Uber formal unentscheidbare satze der principia mathematica und verwandter system i. Monatschefte Math. Phys. 38, 173–198
48. Goldberg, D.: Genetic Algorithms in Search Optimisation and Machine Learning. Addison-Wesley, Reading (1989)
49. Grassberger, P.: Toward a quantitative theory of self-generated complexity. International Journal of Theoretical Physics 25, 907–938 (1986)
50. Harel, D.: Statecharts - a visual formalism for complex systems. SCP 8, 231–274 (1987)
51. Harel, D.: Algorithmics - The Spirit of Computing, 3rd edn. Addison-Wesley, Reading (2004)
52. Holland, J.: Adaptation in Natural and Artificial Systems. MIT Press, Cambridge (1992)
53. Holland, J.: Emergence - from chaos to order. Oxford University Press, Oxford (2000)
54. Hornby, G.S.: Modularity, reuse, and hierarchy: Measuring complexity by measuring structure and organisation. Complexity 13(2), 50–61 (2007)

55. Johnson, J.: Hypernetworks for reconstructing the dynamics of multilevel systems. In: Proceedings of European Conference on Complex Systems (November 2006)
56. Johnson, J.: Multidimensional Events in Multilevel Systems, pp. 311–334. Physica-Verlag HD (2007)
57. Kefalas, P., Eleftherakis, G., Kehris, E.: Communicating x-machines: A practical approach for formal and modular specification of large systems. Journal of Information and Software Technology 45, 269–280 (2003)
58. Kefalas, P., Halcombe, M., Eleftherakis, G., Gheorghe, M.: formal method for the development of agent-based systems. In: Plekhanova, V. (ed.) Intelligent Agent Software Engineering. Idea Group Publishing, UK (2003)
59. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proceedings of the 44th IEEE Symposium on Foundations of Computer Science (2003)
60. Kennedy, J., Eberhart, R.C.: Particle swarm optimisation. In: Proceedings of the IEEE International Conference on Evolutionary computation, pp. 1942–1948 (1995)
61. Kolmogorov, A.N.: On the length of programs for computing finite binary sequences. Prob. Info. Transm. 1, 1–17 (1965)
62. Koppel, M.: Complexity, depth and sophistication. Complex Systems 1, 1087–1091 (1987)
63. Kubik, A.: Toward a formalization of emergence. Artificial Life 9, 41–66 (2003)
64. Kung, S.Y.: Digital Neural Networks. PTR Prentice Hall, Englewood Cliffs (1993)
65. Lano, K.: The B Language and Method: A Guide to Practical Formal Development. In: FACIT. Springer, Heidelberg (1996)
66. Lau, C.: Neural networks, theoretical foundations and analysis. IEEE Press, Los Alamitos (1991)
67. Lovbjerg, M., Rasmussen, T.K., Krink, T.: Hybrid particle swarm optimiser with breeding and subpopulations. In: Proceedings of the third Genetic and Evolutionary Computation Conference (2001)
68. Mamei, M., Menezes, R., Tolksdorf, R., Zambonelli, F.: Case studies for self-organisation in computer science. Journal of System Architecture 52, 160–443 (2006)
69. Mamei, M., Zambonelli, F.: Field-based coordination for pervasive multiagent systems. Springer, Heidelberg (2006)
70. Messie, D., Oh, J.C.: Environment organisation of roles using polymorphism. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2005. LNCS (LNAI), vol. 3830, pp. 251–269. Springer, Heidelberg (2006)
71. Milner, R.: A Calculus of Communicating Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
72. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes (i and ii). Inform. and Comput. 100(1), 1–77 (1992)
73. Van Dyke Parunak, H., Brueckner, S.A., Sauter, J.: Digital pheromones for coordination of unmanned vehicles. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2004. LNCS (LNAI), vol. 3374. Springer, Heidelberg (2005)
74. Patel, J., Teacy, W.T.L., Jennings, N.R., Luck, M.: A probabilistic trust model for handling inaccurate reputation sources. In: Herrmann, P., Issarny, V., Shiu, S.C.K. (eds.) iTrust 2005. LNCS, vol. 3477, pp. 193–209. Springer, Heidelberg (2005)
75. Petri, C.A.: Kommunikation mit Automaten. PhD thesis, Institut fuer Instrumentelle Mathematik, Bonn (1962)

76. Philippides, A., Smith, T., Husbands, P., O'Shea, M.: Diffusible neuromodulation in real and artificial neural networks. In: AI Symposium, Second International Conference on Cybernetics, Applied Mathematics and Physics: CIMAF 1999. Editorial Academia (1999)
77. Picard, G., Gleizes, M.P.: Cooperative self-organisation to design robust and adaptive collectives. In: Second International Conference on Informatics in Control, Automation and Robotics (ICINCO 2005), Barcelona, Spain, September 14–17, pp. 236–241. INSTICC Press (2005)
78. Platon, E., Mamei, M., Sabouret, N., Honiden, S., Parunak, H.V.D.: Mechanisms for environmenrts in multi-agent systems: Survey and applications. Auton. Agent Multi-Agent Syst. 14, 31–47 (2007)
79. Ramchurn, S., Huynh, D., Jennings, N.R.: Trust in multiagent systems. The Knowledge Engineering Review 19(1), 1–25 (2004)
80. Ramchurn, S.D., Jennings, N.R., Sierra, C., Godo, L.: Devising a trust model for multi-agent interactions using confidence and reputation. Applied Artificial Intelligence, pp. 833–852 (2004)
81. Randers, J.: Elements of the System Dynamics Method. MIT Press, Cambridge (1980)
82. Ratnasamy, S., Karp, B.: Ght: A geographic hash table for data-centric storage. In: Proceedings of the international workshop on wireless sensor networks and applications, Atlanta. ACM Press, New York (2002)
83. Ronald, E., Sipper, M.: Design, observation, surprise! a test of emergence. Artifcial Life 5, 225–239 (1999)
84. Rouff, C., Hinchey, M., Rash, J., Truszkowski, W., Gordon-Spears, D.: Formal Methods and Agent-based Systems. Springer, Heidelberg (2006)
85. Ryan, A.: Emergence is coupled to scope, not level. Nonlinear Sciences (2007)
86. Sawyer, R.K.: Simulating emergence and downward causation in small groups. In: Proceedings of the Second International Workshop on Multi-Agent Based Simulation, pp. 49–67. Springer, Berlin (2001)
87. Di Marzo Serugendo, G., Gleizes, G., Glize, P.: Self-organisation and emergence in multi-agent systems. The Knowledge Engineering Review 20, 165–189
88. Shalizi, C.: Causal Architecture, Complexity and Self-Organization in Time Series and Cellular Automata. PhD thesis, University of Michigan (2001)
89. Shalizi, C.R., Crutchfield, J.P.: Computational mechanics - pattern and prediction, structure and simplicity. Journal of Statictical Physics 104, 819–881 (2001)
90. Shalizi, C.R., Shalizi, K.L.: Optimal non-linear prediction of random fields on networks. Discrete Mathematics and Theoretical Computer Science, 11–30 (2003)
91. Shalizi, C.R., Shalizi, K.L.: Blind construction of optimal nonlinear recursive predictors for discrete sequences. In: Chickering, M., Halpern, J.J. (eds.) Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference. AUAI Press (2004)
92. Silberstein, M., McGeever, J.: The search for ontological emergence. The Philosophical Quarterly 49(195), 201–214 (1999)
93. Spivey, J.M.: The Z notation: a reference manual. Prentice-Hall, Englewood Cliffs (1989)
94. Sudeikat, J., Renz, W.: Toward requirements engineering for self-organising multi-agent systems. In: Proceedings of the First IEEE International Conference on self-adaptive and self-organising systems (SASO 2007), pp. 299–302 (2006)
95. Sudeikat, J., Renz, W.: Building complex adaptive systems: On engineering self-organising multi-agent systems. In: Application of complex adaptive systems. IDEA (2007)

96. Sudeikat, J., Renz, W.: Toward systemic mas development: Enforcing decentralised self-organisation by composition and refinement of archetype dynamics. In: Engineering Environment-Mediated Multiagent Systems (EEMAS 2007). LNCS. Springer, Heidelberg (2007)
97. Tononi, G., Sporns, O., Edelman, G.M.: A measure for brain complexity: Relating functional seggregation and integration in the nervous system. PNAS 91, 5033–5037 (1994)
98. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proc. Lond. Math. Soc. 42, 230–265
99. Varela, F.: Principles of Biological Autonomy. Elsevier, New York (1979)
100. Vigano, F., Fornara, N., Colombetti, M.: An event driven approach to norms in artificial institutions. In: Boissier, O., Padget, J., Dignum, V., Lindemann, G., Matson, E., Ossowski, S., Sichman, J.S., Vázquez-Salceda, J. (eds.) ANIREM 2005 and OOOP 2005. LNCS (LNAI), vol. 3913, pp. 142–154. Springer, Heidelberg (2006)
101. Weyns, D., Holvoet, T.: A formal model for situated multi-agent systems. Fundamenta Informaticae 63(2–3), 125–158 (2004)
102. Weyns, D., Vizzari, G., Holvoet, T.: Environments for situated multi-agent systems: Beyond infrastructure. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2005. LNCS (LNAI), vol. 3830. Springer, Heidelberg (2006)