# System of Systems design verification: problematic, trends and opportunities

Mustapha Bilal, Nicolas Daclin, Vincent Chapurlat

# System of Systems design verification: problematic, trends and opportunities

M. Bilal[1], N. Daclin[1] and V. Chapurlat[1]
[1] LGI2P, Laboratoire de Génie Informatique et d'Ingénierie de Production, ENS Mines Alès, Parc scientifique G. Besse, 30035 Nîmes Cedex 1, France.
{Mustapha.Bilal, Nicolas.Daclin, Vincent.Chapurlat}@mines-ales.fr

**Abstract.** System of Systems (SoS) Engineering (SoSE) requires to be able to model and to argue the quality of the modeled solution, thanks to various objectives prior to any other efforts. This paper presents and discusses the development of an approach to support SoSE activities and particularly to achieve SoS modeling and verification. First, requested models are identified and illustrated here on Virtual Enterprise domain (VE). Second, it is proposed to merge two complementary verification approaches, formal proof and simulation. This allows us to ensure particularly the stability, integrity and control expectations of the proposed SoS solution, and must encompass particularly three main SoS characteristics chosen here that can impact SoS stability, integraty and controllability. These characteristics are connectivity, particularly subsystems' interoperability abilities, evolution and emergence of behaviors and properties which are due to the subsystems' interactions when fulfilling the SoS operational mission. For this, a formal properties specification and proof approach allow the verification of the adequacy and coherence of SoS models with regard to these characteristic and to stakeholders' requirements. Then, simulation based on Multi Agents Systems (MAS) allows the execution of the architectural model of SoS. This allows to detect potential emergent operational scenarios and then to obtain an approached behavioral model of the SoS. This MAS is enriched by concepts and mechanisms allowing to evaluate some criteria to facilitate and guide the identification of such operational scenarios.

**Keywords:** System of Systems, interoperability, System Engineering, Verifications, Formalization, Emergence, properties proof, Virtual Enterprise, Multi-Agent System.

## 1 Introduction

The concept of complex system is defined but also often seen as a little limited by several authors when considering large and heterogeneous systems involving other called complex, technical as sociotechnical and interacting systems. The notion of System of Systems (SoS) has been then introduced. Indeed, these ones, such as

Virtual Enterprise (VE), military coalition forces or even crisis management system, have particular characteristics. Moreover, SoS design is distinguished from classical system design [1]. In SoS design it is required to focus designers' attention on interfaces to design, and that to allow subsystems to improve interoperability.Therefore, SoS Engineering (SoSE) remains today an open issue.

In this paper, we define and focus first on such characteristics. Second, we propose a conceptual and tooled approach allowing designers to model a SoS and to analyze the resulting models taking into consideration some of these particular but expected characteristics of a SoS. Presented here, the first phases of an ongoing research project mixing modeling concepts coming from System Engineering, Enterprise modeling, formal verification and advanced simulation techniques.

## The paper is structured as follow. The problematic, related to the modeling and verification issues for SoS Engineering (SoSE) and expected results are presented in section 2. The proposed SoSE approach is introduced in section 3. Finally, section 4 concludes this paper, drawing the orientation of future works.2 Problematic and expected results

### 2.1 SoS: concept and definition

The short literature review presented below allows us to fix the most relevant characteristics of a SoS and its subsystems to study, in order to help designers building SoS models and assuming qualities of the modelled solution. Therefore, a SoS is seen as a combination of systems (subsystems) together to fulfill some kind of capability that a system alone cannot fulfill. It can be considered as a complex system [2]. Furthermore, [3] and [4] mention the following characteristics allowing to distinguish SoS from large and complex but monolithic systems:

- *Operational Independence of the Elements: SoS is composed of subsystems which are independent and useful in their own right.*
- *Managerial Independence of the Elements: The subsystems are separately acquired and assembled but maintain a continuing operational existence independent of the SoS.*
- *Evolutionary Development: The SoS does not appear fully formed. Its development and existence is evolutionary with functions and purposes added, removed, and modified with experience.*
- *Emergent Behavior: The SoS performs functions and carries out purposes that do not reside in any subsystem taken isolated but reside in the various interactions between these subsystems. The principal purposes of the SoS are fulfilled by these behaviors considered here then as emergent behaviors.*
- *Geographic Distribution: The geographic extent of the subsystems is large. Large is a nebulous and relative concept as communication capabilities increase, but at a minimum it means that the subsystems can readily exchange only information and not substantial quantities of mass or energy.*

- *Connectivity: To enable the SoS, subsystems are capable of building links among their interfaces and destroying them dynamically. The SoS places a huge reliance on effective connectivity in dynamic theatres of operations.*
- *Diversity: The SoS can only achieve its higher purpose(s) by leveraging the diversity of its constituent systems.*

Some SoS characteristics (autonomy, belonging, diversity and geographic distribution) are well defined and several works are developed in terms of methodology and tools about these ones [5]. Therefore, the here proposed work takes an interest in three characteristics: Connectivity, Evolution and Emergence.

### 2.2 SoS Engineering problematic

It is admitted that SoS Engineering (SoSE) can be distinguished from System Engineering (SE) [1][6]. Indeed, a SoS results essentially from assembling and interfacing of, in most cases, existing systems in order to fulfill a specific mission (to provide goods and services in agreements with stakeholders' requirements). However, these subsystems must remain independent and have to remain capable of achieving their own mission while SoS is existing. Therefore, they are selected and involved under various conditions and constraints, particularly their interoperability and performances, that have to be characterized prior the assembling. Indeed this assembly establishes various interactions between the subsystems. In this context, interoperability takes on its full meaning when considering these interactions that make these subsystems able to work together. On the one hand, the interactions between subsystems are expected in order to allow to the SoS to fulfill its mission.On the other hand, these interactions imposes to have interfaces of various types: technical (*e.g.* software), organizational (*e.g.* communication rules), human/machine (*e.g.* touchscreens) or logical at a high level of abstraction (*e.g.* resource utilization). Therefore, designers' attention has to be then concentrated on interfaces-to-design in order to ensure the connectivity. Furthermore, SoS should be able to evolute, and by consequence, this evolution, in parallel with the various interactions between the subsystems, can be at the origin of emergent behaviors and properties that remain not easy to identify and can be considered eventually as beneficial or damaging.

Various properties, such as proposed in [7], characterizing the SoS cannot be directly deduced and linked to the set of the properties which characterize separately all the subsystems. In the same way emergent behaviors remain, by definition, not easy to detect in a simple and efficient way. The SoS complexity, the determination of potential interactions and the large number of behavioral scenarios, cannot be totally explored and analyzed. Therefore, several design verification techniques can be used with more or less good results from informal to formal ones in order to keep the SoS characteristics maintained all over its life cycle. Moreover, these approaches and methods are largely used to help detecting errors or mistakes during design activities. Hence, avoiding drawbacks in case of SoS design can be solved by defining an efficient SoS model and adopting model verification techniques of various types.

To synthesize, SoSE process evokes a decision when assembling subsystems. It requires modeling and verification techniques and tools. Therefore, a first barrier is selecting subsystems considering and checking their capabilities and abilities to be/to stay interoperable, and to optimize the impact of a set of properties (functional, behavioral, ilities[1], constraints[2] and performance expectations), for a more or less long time while SoS has to fulfill its mission. Defining the requested interactions, the needed interfaces and, finally, the global architecture of the SoS is a second barrier. Defining metrics and verification or validation techniques allowing proving and checking the same set of properties mentioned previously, that affect the so-called *analysis perspectives* is a third barrier. The Analysis perspectives are defined as:

- *Stability*: is the quality that reflects the ability of a system to maintain its viability (it characterizes the relationship between system's structure and its cohesion).
- *Integrity*: characterizes the relationship between system's behavior and its consistency.
- *Controllability (performance)*: is the quality that reflects the system's ability to achieve its mission (it characterizes the relationship between the functions to be performed by the system and the given service's compliance)

These three barriers have to be treated keeping in mind (1) that a design process involves various disciplines, (2) the size and expected characteristics (connectivity, evolution and emergence) that might impacts the set of properties, and (3) the sociotechnical nature of the requested SoS. The next section presents why SE is not sufficient for SoS to respond properly to our problematic.

### 2.3 SoSE needs

Some authors assume that SE proposes sufficient principles and processes, suitable for the SoSE [8]. However, according to the specificity of a SoS (size, complexity, characteristics) and the *analysis perspectives* we chose, SoSE induces stronger effort for designers.

First, **requirements engineering** activities for a SoS (functional as non-functional) are more complex. In addition to the classical "-ilities" such as reliability, maintainability, availability etc. , new "-ilities" such as interoperability, flexibility, adaptability and composeability are imposed during SoSE. Therfore, designers need an enriched requirement model which includes these new "-ilities".

Second, choosing and assembling the subsystems, which are able to provide requested capabilities/capacities and they respect **model based system engineering** principles, requires having an adapted modelling languages in order to achieve SoS modelling and verification expectations. Due to the specificity of a SoS, some current available modelling languages (*e.g.* behavioral) remain not sufficient for embedding or representing chosen SoS characteristics. Moreover, existing modelling languages do not consider emergent phenomena. Hence, a new behavioral modelling language has to be defined; here based on the enrichment of existing ones. Furthermore, an architectural model (functional and organic) must be proposed allowing rendering SoS architecture characteristics. The challenge is then to formalize the modelling

[1] Refers to the non-functional requirements such as maintainability, safety, security etc.
[2] Legal, deployment, implementation, etc.

languages (ML) that can be used in order to allow to model and to assess the interactions between subsystems. ML must permit to design requested interfaces allowing managing these interactions without inducing huge modifications or dysfunction of each subsystem. These interfaces can be of various types: technical (respecting general standards of physical interconnections of technical systems), physical (hardware), informational (knowledge, information and data exchange protocols), organizational (separation process public/private, protocols and rules of organization, control, taking responsibility, delegation, etc.) or HMI (human machine interface). These interfaces allow designers to ensure the necessary *interoperability* of subsystems [9].

Third, SoS model must allow to check if the modelled solution respects the modelling systems and stakeholders requirements [10]. Indeed, ML must also allow designers to attest that the SoS model is well constructed, well-formed and coherent with these requirements. This has to be done by **verifying**, interpreting and analyzing the obtained SoS models through various methods (formal and semi-formal ones). Formal methods [11] mathematically reasons (proves) the correctness of a given design and the systems specification as well. Formal model verification will allow designers to establish and justify that the models represents accurately the SoS system. As a result, when stating that a system has been formally verified, there should be a detailed explanation of what was formalized and what properties where verified. However, it has been many years since the formal methods are being used but their integration into the industry is still limited, mainly due to state-space explosion problem and the need of a significant knowledge in advanced mathematics. Model Checker is a formal method used for verification of systems. For instance, on the one hand, applying Model Checking techniques [12] needs to describe the system with a formalized modeling language, to formalize the properties by a specification language and to apply a deductive algorithm or calculus for the verification. On the other hand, simulation can be also used simultaneously with formal proof techniques. It is an easy technique that can be automated and is very scalable. It helps to study designs in its early stages. Furthermore, the simulation can ensure a partial validation of a model.

The effort to be made in order to deal with the various issues related to the SoSE process has been presented in this section. The following section shows that to perform SoSE process, we need (1) a set of ML to be identified (2) a set of concepts, an architecture framework –that can be handled by using these ML and (3), to attend to activities in order to ensure and verify the quality of the design, its adequacy and feasibility by merging, in a complementary way, the formal properties specification and proof approach with the simulation approach whatever the size of the SoS and the emergent behavior that might be produced .

## 3 Proposal of SoS engineering (SoSE) approach

To perform SoSE process, SoS modeling and verification concepts have to be identified. Starting with the **modelling phase**, two packages of concepts are requested: *SoS modeling concepts* package and *SoS model management* package. First, from SE point of view, *SoS modeling concepts* package is decomposed into environmental, functional, organic, behavioral and requirement concepts. Second, *model*, *refinement principle* and *view* are categorized into *SoS model management* package. This package allows us to manage the *SoS modeling concepts* package. Environmental concepts group the *context, enabling system, resource, flow* and *stakeholder*. Functional concepts group the *function, construct (parallelism, sequence, iteration, etc.), flow (data, energy* and *material), item, interaction (*characterized by *attributes – Time, Shape* or *Space), resource, effect* and *risk*. The organic concepts group *the interaction, component, link, interface, resource* and *technical indicator (performance, constraint, "-ility"*). The behavioral concept groups at least *component, configuration, transition, capacity, capability, operational scenario, interaction, risk* and *effect*. The requirement concepts group the *role, need, requirement, capability, activity, process, capacity, operational scenario, functioning mode, interaction, life cycle* and *system/subsystem configuration*.

After fixing the most relevant concepts in the **modelling phase**, we present in the following paragraph the relationships phase or what we call the **bonding phase**. In this phase, a definition of some concepts is given with the relationships between each other.

Each stakeholder has concerns and *needs* (functional and non-functional) that are expected to be met by the SoS. For example, in a virtual enterprise (VE) [13], seen as a SoS, the company's stakeholders can be: employee, supplier, community, owner, investor, government, etc. Each one has its own *needs* and concerns (e.g. owners'/companies' needs profitability, longevity, market share, market standing, succession planning, raising capital, growth, social goal, etc.). These *needs* are transformed into a set of *requirements* to be satisfied by the SoS. A requirement modelling language is used in order to formalize the needs into requirements (e.g. –SBVR, natural language, formal language). Moreover, the requirements are classified into various categories (functional, operational, performance, human factors, "-ilites", constraint, interfaces etc.).

Once the *Requirements* are well defined, they are evaluated by some *technical indicators*: criticality, safety, security, interoperability, maintainability, availability, adaptability, flexibility, ilities, performance, constraints and many others. The *stability*, *integrity* and *control* are the **analysis perspective** (basic principles) which we consider in our research. The *technical indicators* are verified in order to determine how they affect the analysis perspectives of the SoS. This will allow to optimise their impact. For example and in the context of VE, the performance is strongly related to three main types: delay, quality and cost. The performance indicator is high: (1) when the VE is capable of respecting the time constraints (accomplishing a mission in a given interval of time), (2) when the quality of the accomplished mission is high and (3) when the cost is low. Moreover, the stability of a VE is the capability of executing the mission/objective whatever the internal/external changes. The integrity is a concept of expectations' and outcomes' consistency in order to keep unambiguous position in the mind of various enterprises forming the VE. Therefore, the integrity here is described as the state of being whole, complete and always in perfect condition.

The *model* is a concept on which the *technical indicators* are described. A SoS is modeled throughout subsystems models. These models have unknown (or partially known) *capacities/capabilities* and *performances*. Their *interactions,* that allow them to fulfill their mission and to get connected to other subsystems or with the environment, remain not clearly identified and modeled. Therefore, this imposes to have a clear architectural model. This architectural model will allow to the subsystems models to be represented with the same formalism and a same level of detail.

As stated in the previous section, some *modeling languages* already exist, each with their strengths and weaknesses, in order to meet SoS design Verification and Validation (V&V). In this way, three approaches can be envisaged for our purpose: (I) the modeling language (ML) is fully adapted and can be directly used, (II) the ML partially covers the concern and it is required to extend it and (III) no ML related to the concern exists and it is necessary to develop new model.

As far as requirements models are concerned, existing ones are adapted to model stakeholders' and subsystems' requirements. They enable concurrent engineering processes to work more efficiently through models and they give a concise picture of the boundaries and constraints that it is expected to operate within a large and complex systems like the SoS.

An environmental model has to be proposed. It should contain the stakeholders, the context and the subsystems. However, a global behavioral model is difficult to build due to the connectivity and interoperability which are a major reason behind the appearance of emergence. Thus, a behavioral model based on interaction and effects models has to be proposed.

A behavioral model reproduces the behavior of the SoS. Due to the dynamicity in the SoS, the behavioral model will never be a global model but it will be able to cover a wide range of behaviors (including the emergent ones) through a simulation technique. The approached behavioral model to build will be a description of how the subsystems will interact together, with the actors and with any entity which is out of the SoS's boundary and from here comes our proposal to build this model based on interactions and effects models, no more described in this paper.

An interaction model is proposed to describe how subsystems have to exchange flows. Various interfaces are then defined in order to ensure the necessary connectivity (interoperability) of the subsystems respecting or, if needed, managing reverse effects due to these interactions. These interfaces establish the *links* (e.g. protocol, synchronization, collaboration, delegation rules, etc.) which exist between SoS subsystems. These links transport a *flow* (continuous, discrete or hybrid). Moreover, this interaction model should include the *effects,* which induce some kind of *risks* (e.g. technical, managerial, human, financial etc.).

In our case, the following table (see Table 1) presents all models used to ensure SoS modeling. These models are consistently related to the concerned views. One of the main differences between the architecture of a complex system and the SoS architecture is its dynamic in its reconfiguration [14]. Therefore, an architecture for the SoS has to be proposed. Proposing an architecture is to define the fundamental organization, its subsystems, the interactions between these subsystems, the environment, and the principles which guide SoS design and evolution.

Once the concepts and the models are well identified, a choice of the modeling languages has to be taken. In that sense, SysML seems allowing requirements description through its Requirement Diagram and interaction description (interoperability) through its Activity Diagram, Sequence Diagram and State Machine Diagram. However, SysML is considered as semi-formal modeling language and it remains too limited for building other models. Therefore, it has to be enriched and formalized in order to permit formal checking and to facilitate simulation. For example, the Requirement Diagram of SysML allows us to collect and organize all the textual requirements of the subsystems. However, a SoS has its own characteristics/requirements to be considered (connectivity, evolution and emergence). These new requirements evoke some new concepts that cannot be modeled by SysML (coordination of communication between the subsystems, adaptability, confidence etc.).

Verifying the SoS model, whatever may be its size and the complexity of its subsystems, is not being yet fully discovered by the research. All the verifications that have been done concern only specific application domains [15]. We consider that formal verification is not efficient since: (1) it is not sufficient for the challenge to establish SoS Integrity, stability and performance such that they fulfill valid requirements of their users with expected quality and are constructed in cost effective way and (2), pure formalization and verification can only prove a correct relationship between formal specifications and implementations but cannot prove that the SoS meet valid requirements. Therefore, it is important to have a mathematical formalization of engineering concepts. Moreover, engineering concepts in systems are mostly complex and abstract and they are difficult to define properly, to understand and to justify.

**Table 1** SoS models and views

| **views** | **SoS models** |
|---|---|
| Functional, logical and physical views | Architectural model (including physical, functionnal, interface and interaction models) |
| Requirements view | Requirement model |
| External view | Environmental model |
| Behavioral view | Behavioral approached model |

The verification methodology presented in this paper is based on the use, in a complementary way, of formal proofs (Model Checker) and simulation techniques through the Multi-Agent Systems (MAS). However, using Model Checker for verification requests to have a global and deterministic behavioral model of SoS, but this model cannot be described when facing emergence phenomenon. Therefore, the use of Model Checker should be at a definite instant of the simulation. The methodology to follow is summarized in the following steps:

1. A scenario is initialized (parameter models),
2. Some scenarios are emerged,
3. The simulation is stopped at a time “t”,
4. Information (the set of properties: functional, behavioral, ilities, constraints and performance expectations) are retrieved from agents,
5. An agent responsible of using Model Checker will verify this set of properties and their impact on the analysis perspective.

The choice of MAS refers to the fact that it is natural and an effective solution to deal with complex situation in distributed environments [16], it allows modeling and simulating the parallel evolution and the interactions of various complex subsystems independently and it can answer to the individual failure of one of the elements without degrading the whole system. Moreover, it is widely used in various domains (parking [17], biomedical science, crime analysis, environment evaluation etc.).

In our case, agents represent the active entities of the SoS (subsystems). However, the subsystems can be of various natures which raises the importance of defining various kinds of Agents, such as: Intentional Agents, Rational Agents and Situated Agents. A BDI technology (Beliefs, Desire, Intention), used in some MAS, allows to model more accurately, the knowledge and rules of behavior to be exhibited by the agents which model each subsystem.

To model the SoS architecture, Multi-Agent models structure is defined according to six dimensions:

- Agent model represent the active entities (subsystems)
- Environment model is where the SoS exists and with which they interact
- Interaction model manages the interaction between the subsystems
- Model Checker model uses formal proof techniques applied to a set of properties
- Evaluation model uses evaluation techniques applied to properties translating analysis perspective expectations (stability, integrity and controllability)
- Organization model defines constraints and rules on the interaction model

As stated previously, the simulation will allow emerging some scenarios. However, an emergence can be beneficial, harmful, or neutral in its effect. It is the primary mechanism for both success and failure in SoS. Therefore, we need to determine what the harmful and beneficial ones are.

In our research, we consider two types of emergence out of four proposed by [18]. Detecting and filtering the emergent behaviors is achieved through some criteria: the emergence need to be observable at some level, novelty, coherence irreducibility (a complete account of an entity will not be possible at lower levels of explanation and which has novel properties beyond prediction and explanation), interdependency between levels, non-linearity, plausibility and credibility. Further details about each criterion will not be shown in this paper due to the lack of space.

## 4 Conclusion and prospects

This paper has introduced the importance of SoS design model verification through the complementarity between the formal proof techniques and simulation in order to verify SoS design model and to detect errors and emergent behaviors (which are due to interaction/connectivity and interoperability) in early stages of architectural and interfaces design. Moreover, we have seen how a set of properties can impact the analysis perspectives (stability, integrity and control - performance) of a SoS.

The SoSE approach presented consists on identifying the models to verify inside a SoS, identify the concepts, choose the modeling language and enriched it, and then start to verify these models by a mathematical formalization and by an adequate verification tool simultaneously with the simulation.

We aim to develop a Meta model for SoSE covering architectural model, users' and systems' requirements model, behavioral approached model (including interaction and effect based models) and environmental model, then to propose a mathematical formalization of elements from this Meta model. We are willing to propose as well a repository of expected SoS properties and a verification techniques in order to simultaneously check properties and simulate the approached behavioral model.

## References


[1] S Blanchard, B., J Fabrycky, W.: Systems Engineering and Analysis. 5th edn.(2011)

[2] Chapman, W.L., Bahill, A.T.: Complexity of the system design problem

[3] Maier, M.W.: Architecting principles for systems-of-systems. Systems Engineering 1(4) (1998) 267–284

[4] Stevens Institute Of Technology,Castle Point On Hudson, Hoboken, NJ 07030: Report On System Of Systems Engineering. August 2006

[5] DeLaurentis, D., "Research Foundations," School of Aeronautics and Astronautics, Purdue University, West Lafayette, IN, 2007

[6] Sheard, S. 2006. Is Systems Engineering for "Systems of Systems" Really Any Different? INCOSE Insight, Volume 9 Issue 1, October 2006

[7] Olivier L. de Weck, Adam M.Ross, Donna H. Rhodes, Investigating Relationships and Semantic Sets amongst System Lifecycle Properties (-ilities), third International Engineering Systems Symposium CESUN 2012, Delft University of Technology, 18-20 June 2012

[8] Clark, J.O., "System of Systems Engineering and Family of Systems Engineering from a standards, V-Model, and Dual-V Model perspective," *Systems Conference, 2009 3rd Annual IEEE* , vol., no., pp.381,387, 23-26 March 2009

[9] Mallek, S., Daclin, N., Chapurlat, V.: The application of interoperability requirement specification and verification to collaborative processes in industry. Computers in Industry 63(7) (September 2012) 643–658

[10] Chapurlat V., UPSL-SE: A Model Verification Framework for Systems Engineering, International journal Computers in Industry, COMIND, Elsevier pub., 2013

[11] Woodcock, J., Larsen, P. G., Bicarregui, J., & Fitzgerald, J. (2009). Formal methods: Practice and experience. *ACM Computing Surveys (CSUR)*, *41*(4), 19.

[12] Clarke, E., Schlingloff, H.: Model checking. Handbook of automated reasoning, Elsevier, Amsterdam (2000)

[13] Camarinha-Matos, L.M., Afsarmanesh, H.: The Virtual Enterprise Concept.(1999)

[14] T. Saunders, et al, in "United States Air Force Scientific Advisory Board Report on System of Systems Engineering for Air Force Capability Development" SAB-TR-05-04, July 2005

[15] Drusinsky, D., Michael, J.B., Shing, M.t.: Behavioral Modeling and Run-Time Verification of System-of-Systems Architectural Requirements. (2004)

[16] R. Khosla, T. Dillon, "Intelligent hybrid multi-agent architecture for engineering complex systems," Proceedings of the 1997 IEEE international Conference on Neural Networks, vol. 4,pp. 2449-2454

[17] Bilal, M., Persson, C., Ramparany, F., Picard, G., & Boissier, O. (2012, June). Multi-Agent based governance model for Machine-to-Machine networks in a smart parking management system. In *Communications (ICC), 2012 IEEE International Conference on* (pp. 6468-6472). IEEE. Ottawa, Canada.

[18] Jochen Fromm, Types and Forms of Emergence, Kassel University Press, 2005