Electronic Communications of the EASST
Volume 17 (2009)

Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2009 (WowKiVS 2009)

Systematically Engineering Self-Organizing Systems: The SodekoVS Approach

Jan Sudeikat, Lars Braubach, Alexander Pokahr,
Wolfgang Renz and Winfried Lamersdorf

12 pages

Guest Editors: M. Wagner, D. Hogrefe, K. Geihs, K. David
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Systematically Engineering Self-Organizing Systems: The SodekoVS Approach

Jan Sudeikat[1], Lars Braubach[2], Alexander Pokahr[2], Wolfgang Renz[1] and Winfried Lamersdorf[2]

[1] sudeikat; renz @informatik.haw-hamburg.de
Multimedia Systems Laboratory,
Hamburg University of Applied Sciences, Germany

[2] braubach; pokahr; lamersd@informatik.uni-hamburg.de
Distributed Systems and Information Systems,
Computer Science Department, University of Hamburg, Germany

Abstract: Self–organizing systems promise new software quality attributes that are very hard to obtain using standard software engineering approaches. In accordance with the visions of e.g. *autonomic computing* and *organic computing*, self–organizing systems promote self-adaptability as one major property helping to realize software that can manage itself at runtime. In this respect, self-adaptability can be seen as a necessary foundation for realizing e.g. self* properties such as self–configuration or self–protection. However, the systematic development of systems exhibiting such properties challenges current development practices. The SodekoVS project addresses the challenge to purposefully engineer adaptivity by proposing a new approach that considers the system architecture as well as the software development methodology as integral intertwined aspects for system construction. Following the proposed process, self-organizing dynamics, inspired by biological, physical and social systems, can be integrated into applications by composing modules that distribute feedback control structures among system entities. These compositions support hierarchical as well as completely decentralized solutions without a single point of failure. This novel development conception is supported by a reference architecture, a tailored programming model as well as a library of ready to use self–organizing patterns. The key challenges, recent research activities, application scenarios as well as intermediate results are discussed.

Keywords: Distributed Systems, Self–Organization, Decentralized Coordination

1 Introduction

Innovative application scenarios are often very demanding with respect to the desired features the systems should exhibit. Examples include groups of small low-cost satellites that are able to perform a mission in concert,[1] underground urban transport systems consisting of many small

[1] e.g. the NASA Autonomous NanoTechnology Swarm (ANTS) / Prospecting Asteroid Mission (PAM) project: http://ants.gsfc.nasa.gov/pam.html

autonomous vehicles, each responsible for reaching an individual target area[2] and the monitoring and automatic reconfiguration of server farms in case of changing environments (e.g. [KLT+08]). One requirement here is that, because single entities of the systems may fail at any point in time, e.g. a satellite runs out of energy or a server has a hardware failure, it is of vital importance that those breakdowns do not harm the overall system functionality.

All mentioned examples have in common that they assume a decentralized infrastructure consisting of a multitude of autonomous entities which have to interact in order to perform the intended functionalities. This high number of entities as well as the demand for autonomous behaviour in combination with coordinated actions requires novel software concepts. In addition, the ability to systematically construct these kinds of systems is also an important aspect, which is not well-supported until now.

In the new DFG-funded research project "*Selbstorganisation durch Dezentrale Koordination in Verteilten Systemen*"[3] (SodekoVS) the aforementioned problems will be tackled by utilizing nature-inspired design paradigms. These provide coordination strategies to equip software architectures with adaptability and robustness, based on decentralized self-organization principles. The project mainly aims at making these strategies software technically applicable. Basis for this exploitation are a newly conceived generic reference architecture as well as an adapted development methodology for the systematic construction of such systems. The project shares parts of its vision with research directions like autonomic and organic computing, and tries to employ ideas from these areas wherever possible. Nonetheless, the focus of this project is more on the engineering area and thus more concerned with bringing together standard software-engineering approaches with nature-inspired paradigms and allow for exploiting them in a generic reusable way.

This paper is structured as follows. In the next section we discuss the concept of self–organization and motivate the utilization of this concept as a tool to engineer system adaptivity. In the following section 3, the imminent research challenges are presented that have to be addressed to enable the purposeful engineering of self–organizing processes. Application scenarios (section 4) are outlined before we conclude and give prospects for future work.

2 Self–Organization in Software

The systematic utilization of self–organizing dynamics represents an important prerequisite for being able to construct challenging applications and equip them with the ability to manage themselves, allowing for desired properties like adaptability and robustness. Here, we will introduce the notion of self-organization and outline the current state of support for its software technical exploitation.

2.1 Self–Management via Self–Organization

The term *self–management* has been coined to describe software systems that exhibit *adaptivity*, i.e. that adjust their configuration at run–time [MWJ+07]. This view follows the black–box definition of adaptivity by [Zad63] that characterizes adaptivity as the ability to respond to inputs

[2] e.g. the CargoCap Project: http://www.cargocap.de/

[3] Self–Organisation by Decentralized Coordination in Distributed Systems

with *appropriate* outputs, where the appropriateness of responses is subject to system observers.

The establishment of self–management requires the introduction of *control loops*. Systems are to be monitored, the appropriateness of perceived states is to be evaluated and actions are to be selected and enacted that adjust the systems configuration. Several research efforts, most prominently the *autonomic* [HM08] and *organic* [SOR07] computing initiatives, have established frameworks[4], design patterns [SOR07] and architectures [RMB+06] that facilitate the creation of *managing* elements that automate these activities, i.e. establish closed control loops.

The term *self–organization* originates from the description of physical, biological and social phenomena, where global structures arise from the local interactions of individuals (e.g. particles, cells, agents, etc.) [SGK06]. Self–organizing system operate completely decentralized, self-actuate the rise of structures and behave adaptive while being subject to perturbations. These phenomena can be observed when system elements act autonomously and their activities influence each other mutually. These influences enable *decentralized coordination* by distributed control loops. A canonical example is the so–called *Ising*-Model[5] that explains ferro–magnetization as a cooperative effect of micro–magnets, so–called *spins*. The magnetization of these influence each other and when individual spins are able to adjust their heading, these align to a coherent magnet field. Self–organization is distinct from *emergence*, which describes the establishment of higher level artifacts that are irreducible to the constituent system entities. Self–organization is often a prerequisite to emergent properties, but a distinct concept [DH04].

Self–Organized phenomena are a threat and at the same time an opportunity for the development of adaptive applications. Self–organization threatens todays applications as it may be introduced unintendedly, e.g. discussed in [Mog05]. It is non–trivial to foresee the mutual, transient influences among autonomous, distributed entities that may lead to cooperative effects. Influences can be caused by direct perceptions or indirect dependences, i.e. due to shared resources. While superior management entities can be used to dampen these effects, approaches to consider and plan for self–organized phenomena are indispensable. The purposeful utilization of self–organization is an active field of research [SGK06] and has been proven useful as software design paradigm for self-managing systems in several application domains [PB04, NT04, SR08a].

The utilization of this paradigm promises to *weave* adaptive properties into applications, i.e. to inherently manifest adaptive properties. By controlling the collective adjustment of component configurations, the coherent adjustment of the system itself can be conceived. This design approach is attractive as it allows to distribute problem solving computations among system entities. The absence of dedicated managing entities fosters achieving robustness, scalability and failure recovery. However, developers need to be aware that self–organized mechanisms often only provide near–optimum solutions and can be outperformed by centralized managers that have complete knowledge. The distribution of problem solving power also requires background processing in the coordinated elements.

2.2 Developing Self–Organizing Systems

Developers of self–organizing applications face the dilemma how to design the the constituent entities (micro-level) in order to establish the intended system–wide properties (macro-level).

[4] e.g. the Autonomic Computing Toolkit: http://www.ibm.com/developerworks/autonomic/overview.html

[5] e.g. see http://ccl.northwestern.edu/netlogo/models/Ising for an example simulation model

Prominent tools for the development of self–organizing applications are field–tested *coordination strategies* and simulation–based bottom–up *development procedures*, e.g. reviewed in [SR08a]. These provide practical guidance on how to conceive and revise non–linearly coordinated system properties.

Strategies for the coordination of autonomous entities typically take inspiration from natural, e.g. physical, biological or social, self–organizing systems that serve as *design metaphors* (e.g. catalogued in [MMTZ06]). Following these designs, entities (agents) communicate via so–called *Decentralized Coordination Mechanisms* (DCM) [DH06a]. For example, the trail formation of ant during *ant foraging* can serve as design inspiration and is realized by distributing *digital pheromones* [PB04]. The nature–inspired metaphors provide reusable patterns of control loops [SR08a, SR07b] that can be realized by allowing entities to collectively adjust their local behaviors. Individual adjustments are based on *information flows* among entities. Decentralized coordination mechanisms guide their realization by direct or environment–mediated interactions that provide dynamics for spreading and decaying information. E.g. digital pheromones evaporate, therefore allowing old, i.e. not reinforced, information to die out.

The behavior of self–organizing applications arises from the coaction of individuals. Therefore, it cannot be directly inferred from the designs of the individual components but it is necessary to observe the effects of the sum of agent interactions [Edm04]. Simulation–based development procedures support the utilization of self–organizing applications that treat development as a sequence of experiments [Edm04].

2.3 Existing Obstacles for Using Self–Organization in Software Systems

The construction of self–organizing applications challenges current development practices. The decision to utilize self–organizing dynamics has impact on the the development procedure and the application structure. The available design patterns primarily serve as design inspirations, and expert knowledge is required to map these ideas to application domains. In order to anticipate the effects of changes within an element type on the globally rising structure, system simulations are required. Simulations need to be conducted repeatedly to ensure that the intended mechanism / metaphor combinations are capable to exhibit the intended dynamics as well as to quantitatively tune implementation parameters [Edm04]. The results are therefore often highly customized algorithms that cannot easily be reused, i.e. adjusted to further application domains.

The systematic design of decentralized self–organizing systems is scarcely supported. The derivation of guidelines and heuristics for the utilization, selection and combination of coordination strategies [PB04] is impaired by a lack of practical modeling approaches and the diversity of available coordination strategies and corresponding simulation / implementation frameworks. Moving towards the purposeful engineering of self–organized dynamics requires to provide self–organizing dynamics as modeling and implementation concepts, to allow their systematic treatment, reuse and exchange, by non–expert development teams.

3 SodekoVS: Toward Engineering Self–Organization

The SodekoVS project aims at providing self–organizing processes as reusable elements that developers can systematically integrate into their application designs. The utilization of self–

organization in software engineering is addressed by providing a *reference architecture* that offers a conceptual framework for the configuration and integration of self-organizing processes (cf. section 3.1). The integration is guided by adjusting methodical development procedures (cf. section 3.2). Following this conception, coordination mechanisms are made available as middleware services (cf. section 3.1.1). A minimal-intrusive programming model (cf. section 3.1.2) allows developers to configure and integrate representations of nature-inspired coordination strategies in their applications. The systematic utilization of these development tools requires support to design, i.e. model, select, combine and refine self-organizing dynamics (cf. section 3.2.1), and to simulate *qualitatively*, at early development stages, as well as *quantitatively*, the resulting application prototypes (cf. section 3.2.2).

3.1 Architectural Challenges

The aim of the reference architecture is to provide a conceptual framework for the integration of miscellaneous coordination strategies into application designs. It guides the interplay of application dependent component functionalities and their coordination and therefore prepares a constructive approach to the utilization of self–organization. This coherent environment for the definition and implementation of self–organized coordination provides a framework to the assessment, comparison and combination of different self–organizing mechanisms. In addition, the architecture is devised to support the systematic development of self–adaptive applications (cf. section 3.2). The architectural conception makes minimal assumptions on the elements that are to be coordinated and therefore allows to supplement also specialized management architectures, like controller hierarchies from autonomic computing (e.g. [SOR07, RMB+06]).

3.1.1 Providing Self–Organizing Dynamics as Software Components

The diversity of coordination strategies, description formats and implementation / simulation frameworks complicates the treatment of self–organizing dynamics as reusable, exchangeable software components. The unification of mechanism description and implementation models promises a coherent architectural view–point and implementation environment.

In order to examine the ability to adapt self–organizing dynamics to application specific problems, a library of coordination mechanisms is provided. A generic usage interface allows their reuse and facilitates their software technological treatment. The provided library provides a catalogue of mechanism pattern as reusable components that provide systematic problem–oriented descriptions of mechanisms in an abstract, reusable format. This facilitates the selection and combination of mechanisms.

Figure 1 shows the reference architecture [SR08b] for self–organizing application, which is meant as a coarse blueprint for applications having some self–organization features. It consists of three abstraction layers. The top most application layer contains the standard application functionalities and in addition may have a link to agents, which are mainly responsible for realizing the self–organization characteristics, but may also exhibit other application functionalities. The coordination layer placed below consists of the agents as well as a substrate, which may contain one or more coordination media. Mechanism instances are encapsulated in distinct *coordination media* and are interfaced by coordination components that allow to modify agent states.

This model allows to enact coordination strategies by (1) configuring the component internal information to be exchanged, (2) defining the dynamics of the information flows between components and (3) declaring how individual components adjust their local activities (cf. section 3.1.2). Finally, the execution infrastructure layer is responsible for providing basic services to the coordination layer. These services e.g. include the agent management and execution.

Based on detailed classifications of coordination mechanisms properties [SR08a, DH06a, SGK06] generic environment and interaction models remain to be devised that combine and provide the functionalities that allow coordination media to be configured to resemble the dynamics of established coordination mechanisms (cf. section 2.2).

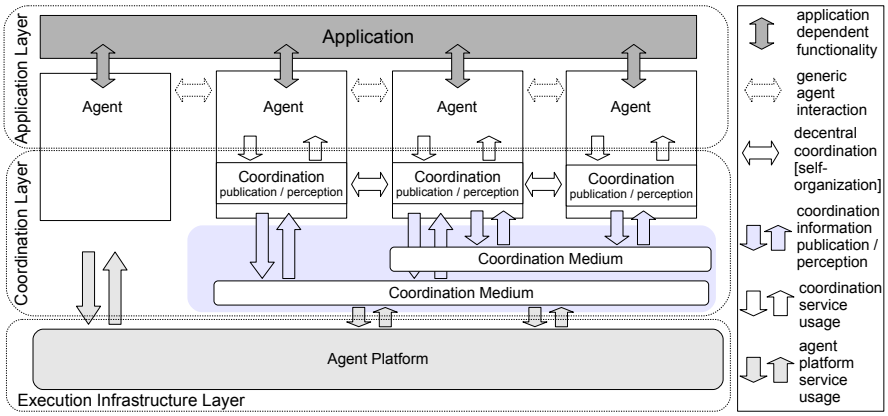

Figure 1: SodekoVS reference architecture, following [SR08b].

3.1.2 Application Integration

Designs of self–organizing applications typically harmonize the coordinated subjects and the means to their coordination, i.e. component and coordination models are tightly coupled. The systematic utilization of the provided coordination media demands conceptual and technological solutions to defining the correct interplay of coordination interactions and application dependent component functionalities. A generic usage interface is required that respects component autonomy and allows the reliable integration of inter–component coordination. The integration interface abstracts from the specific coordination mechanism instances, therefore supporting their arbitrary integration as well as their interchangeability and reconfiguration.

The encapsulation of self–organizing mechanisms and their integration in agent models has found minor attention. In [SPK06, SRR06], adjustments of agent models have been proposed that allow to equip individuals with observer / controller components that enforce self–organizing dynamics by monitoring and modifying agent states. These approaches facilitate the separation of mechanisms to the enforcement of self–organizing dynamics from the application development, i.e. the construction of application dependent agent models.

In [SR08b], an alternative, minimal–intrusive approach has been proposed that allows developers to control the influences of coordination services (cf. section 3.1.1) by annotating agent models. These annotations define which agent internal elements are monitored and effected by coordination services.

This approach promises the separation of coordination and application models. It aims at providing a generic integration interface that abstracts from mechanisms details. Enabling this independence from mechanism implementation details is required to facilitate the coherent definition of coordination strategies and the interchangeable usage of coordination mechanisms.

3.2 Methodical Challenges

The outlined reference architecture allows to resemble nature–inspired self–organization strategies and integrate them in conventionally developed software systems. Supporting the systematic integration by a generally accepted development process is a practical challenge and requires tools to the design and evaluation of self–organizing dynamics. Figure 2 denotes a conceptual view on integrating self–organization. Incremental development activities are supplemented with activities that address the manifestation of self–organizing phenomena (I–V). While developers design the functionality of their applications, they revise the decentralized coordination of component activities in interleaved development activities. Supplements to the *requirements* activities (I) facilitate the description of the intended application dynamics [SR07a]. During *analysis* activities (II), it is examined which instances or combinations of coordination metaphors (cf. section 2.2) can bring by the required adaptivity, i.e. component activity coordination. *Design* activities (III) detail the models of selected coordination strategies and configure the coordination mechanisms that are used for their realization [SR09a]. These activities prepare the *implementation* / integration (IV) of the coordination dynamics within the application software. This is facilitated by a library of mechanism instances to be configured and accessed by a generic usage interface. *Testing* (V) activities are supplemented with the simulation-based validation that component coaction meets the given requirements, i.e. manifests the intended adaptiveness. The detection of variations enforce tuning mechanism parameters (III) and/or coordination redesigns (II).

The major challenges in providing these development activities consist in conceiving practical means to model self–organizing dynamics that facilitate design activities, i.e. coordination strategy selection, combination and refinement (cf. section 3.2.1). The validation that modeled and implemented dynamics meet system requirements is a laborious effort that explicit models of intended dynamics promise to support by tailored system simulations (cf. section 3.2.2).

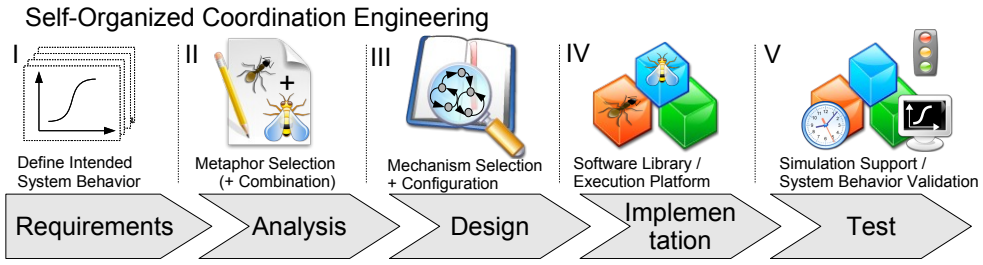

Figure 2: SodekoVS development activities.

3.2.1 Designing Self–Organizing Dynamics by Refining Coordination Strategies

Approaches to model self–organizing dynamics typically rely on sophisticated formalisms that facilitate the derivation of macroscopic system properties from local element specifications e.g.

[RZT07]. Facilitating the *incremental* design of these dynamics requires modeling formalisms that support refinement operations and allows to *align* coordination models with application designs. Systematic utilization of coordination strategies demands that guidelines for the selection and combination of design models can be derived from strategy descriptions.

Distributed control loops have been identified as the driving force of self–organizing dynamics (e.g. in [PB04], cf. section 2) and their explicit modeling promises an approach to relate coordination models to entity designs. In [SR07b, RS08, SR09a] this approach has been proposed to describe the coordination in agent–based applications. The proposed modeling notions take inspiration from the macroscopic modeling level that is adopted by *System Dynamics* (SD) modeling concepts [Ste00]. SD describes systems in terms of system state variables, i.e. accumulative values of system qualities, and causal relations among them which denote the rates of change of variable values. This view–point highlights how system elements mutually influence and therefore facilitate the description of feedback loops. Transferring these *systemic* modeling notions to the semantics of agent–based application designs [SR09a] allows to describe how agent activities influence each other as well as environment properties. Hence, this modeling approach provides an additional view–point on detailed agent designs and the derived models indicate the dynamic system behavior [Ste00].

The approach has been successfully applied to describe the requirements on the adaptivity of self–organizing applications [SR07a]. These descriptions utilize systemic application models and classify the space of possible system configurations, i.e. possible value ranges of state variables. In this respect, adaptivity is described by transitions between types of macroscopic system states.

Coordination strategies provide reusable means to steer the collective reconfiguration of components. Enabling their methodical utilization demands to support their selection, combination as well as the configuration of coordination mechanism implementations. The systematic selection of coordination strategies hasn't been significantly supported in literature. An exception is [DH06b], where coordination mechanisms have been associated to the properties of the macroscopic system behaviors that they are capable to enforce. Coordination metaphors have been classified (e.g. see [MMTZ06]) but unambiguous criteria for the systematic selection remain to be derived. Systemic modeling notions allow to describe coordination strategies. These descriptions particularly facilitate the anticipation of the macroscopic behaviors that result from strategy utilizations either by deriving simulation models or their mathematical treatment. Therefore, this approach promises to facilitate the combination of coordination strategies, e.g. as approached in [SR07b, RS08, DH06a].

These results demonstrate that models of application dynamics can be mapped to agent–oriented design models. The detailed examination of their relation remains future work. It needs to be examined how to derive systemic models from established design notations, possibly supported by tools to maintain the consistency of both modeling levels.

3.2.2 Validation–Support for Self–Organized Applications

Systematic development of self–managing applications demands approaches to validate that systems exhibit the intended characteristics. When the self–management is realized via self–organization, structured procedures are required to to check that the coaction of system elements

gives rise to the expected system behaviors. Basically, two different approaches can be exploited. Formal verification of self–organizing dynamics (cf. e.g. [RZT07]) and systematic simulation studies [Edm04]. Simulation approaches are generally applicable and do not require advanced mathematical skills from developers and hence fit better for mainstream software development.

Self–organizing applications designs can be validated by both *qualitative* as well as *quantitative* simulations. Qualitative simulations (cf. e.g. [GVO06]) allow to check at early development stages that application designs are capable to exhibit the intended system behaviors. Developers derive simulation models from application designs, examine their dynamic properties and when indicated revise application designs. Within the SodekoVS framework, developers will apply simulations to check that combinations of coordination strategies are capable to meet system requirements.

Realizations of coordination strategies (cf. section 3.1.2) need to be simulated to validate their qualitative behavior as well as to quantitatively adjust implementation parameters. The definition of simulation experiments and the interpretation of their results requires manual effort. The SodekoVS framework aims to support and automate these. The ability so simulate applications will be integrated in the coordination middleware service (cf. figure 1). This allows to directly simulate the application code, possibly equipped with mock third party elements, and reduces the effort to repeatedly check the effects of decentralized coordination. The definition of system requirements (cf. section 3.2.1) facilitates the definition of *hypotheses* on the manifested causal relations among system variables that can validated via system simulations [SR09b]. This introduces the *testability* of self–organized phenomena and its practicability and automation will be examined in a prototype implementation of the presented reference architecture (cf. section 3.1).

4 Application Scenarios

In many application areas, challenges exist that can be tackled by employing self-organized dynamics. These challenges are usually induced by the fact that applications need to be deployed in unpredictable and constantly changing environments. Examples of successful self-organizing applications can be found, e.g., in the areas of production control [JB03], ad-hoc/sensor networks as well as robots and unmanned vehicles [MMTZ06]. In all of these settings, self-organization provides advantages compared to traditional solutions, because of the natural robustness and adaptivity of the underlying algorithms.

These algorithms have to be conceived and implemented by trial-and-error in a tedious and manual way. The SodekoVS project strives to simplify and overcome this problem and enable a systematic construction of the desired dynamics. For all phases of the development process, supporting techniques will be provided, which allow focusing on domain relevant factors like costs and capacity utilization. To illustrate this vision and the expected benefits, two example application scenarios will be described in the following, which will be used among others to validate the practical applicability of the SodekoVS approach.

In the area of transportation logistics, domain relevant factors are, e.g., the utilization of trucks and packet delivery time. Among the environmental challenges are the uncertainty about the amount and destinations of future orders, the dynamic occurrence of traffic jams as well as unexpected truck breakdowns. These domain factors and environmental challenges are highly

interrelated. Instead of fighting the connections between factors, the SodekoVS approach focuses on exploiting existing and newly introduced feedback loops to robustly and adaptively balance domain factors as desired. In this respect, self-organized resource/task allocation can be used to balance truck utilization vs. packet delivery time and self-organized routing algorithms ensure that trucks follow nearly optimal routes even in the presence of potential traffic jams [PBS+08].

The management of application within Service–Oriented Architectures (SOA) poses considerable administrative overhead. The reduction of this expense factor is of economic interest. Application elements like databases, application servers and (web) services, are complicated software components that often require manual administration. E.g. in [KLT+08], it has been proposed to save energy (cost) by adaptively switching physical servers. The deployment of application services is also subject to manual adaptation. E.g. when service demands are fluctuating over time and locations, application placements need to be adjusted (cf. e.g. [NT04]). The SodekoVS approach provides coordination media that allow to distribute feedback loops among system entities, i.e. to exchange coordination relevant information that trigger component adjustments. E.g. service brokers may publish service demands and (physical) servers may publish the availability of resources, i.e. the ability to host web services. These information would allow managers of service endpoints to balance service deployments with service workloads. The declarative integration of a comparable coordination strategy is discussed in [SR08b].

5 Conclusions

In this paper it was argued that in many challenging application areas properties like robustness and adaptability are inevitable. In order to provide these properties it is necessary to consider decentralized solutions without a single point of failure. For the coordination of such systems self-organization can be used, but the integration of self-organizing coordination strategies into software systems is a non-trivial activity. Hence, in this paper the vision of a novel development approach for constructing self-organizing systems in a systematical software engineering way was presented. This vision is currently pursued in the context of the SodekoVS project.

Starting point for realizing the vision consists in a new general conceptual framework for building self-organized applications. This framework proposes a generic reference architecture as well as a methodical development support. The reference architecture illustrates which constituents make-up a self-organized application and how these constituents interact, while the development approach shows, which additional activities should be performed in each of the standard development phases. By using the conceptual framework, the current state of the art, the preliminary results of the project as well as the open research questions have been discussed.

Future work concerns the outlined challenges, particularly the realization of the presented coordination architecture, generic coordination environment models and their validation via an integrated simulation support. Moreover, the developed concepts and tools will be used to realize software demonstrators in the outlined application scenarios.

Acknowledgments

SodekoVS is funded by the Deutsche Forschungsgemeinschaft (DFG).

Bibliography

[DH04] T. DeWolf, T. Holvoet. Emergence and self-organisation: a statement of similarities and differences. In *Proceedings of ESOA'04*. Pp. 96–110. 2004.

[DH06a] T. DeWolf, T. Holvoet. Decentralised Coordination Mechanisms as Design Patterns for Self-Organising Emergent Applications. In *Proceedings of ESOA'06*. 2006.

[DH06b] T. DeWolf, T. Holvoet. A Taxonomy for Self-* Properties in Decentralised Autonomic Computing. In *Autonomic Computing: Concepts, Infrastructure, and Applications*. 2006.

[Edm04] B. Edmonds. Using the Experimental Method to Produce Reliable Self-Organised Systems. In *Engineering Self Organising Sytems*. LNAI 3464, pp. 84–99. 2004.

[GVO06] L. Gardelli, M. Viroli, A. Omicini. On the Role of Simulations in Engineering Self-organising MAS: The Case of an Intrusion Detection System in TuCSoN. In *Engineering Self-Organising Systems*. LNAI 3910. 2006.

[HM08] M. C. Huebscher, J. A. McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.* 40(3):1–28, 2008.

[JB03] N. R. Jennings, S. Bussmann. Agent-based control systems. *IEEE Control Systems* 23(3):61–74, 2003.

[KLT+08] R. D. J. O. Kephart, C. Lefurgy, G. Tesauro, D. W. Levine, H. Chan. Autonomic Multi–Agent management of Power and Performance in Data Centers. In *Proc. of AAMAS 2008 – Industry and Applications Track*. Pp. 107–114. 2008.

[MMTZ06] M. Mamei, R. Menezes, R. Tolksdorf, F. Zambonelli. Case studies for self-organization in computer science. *J. Syst. Archit.* 52(8):443–460, 2006.

[Mog05] J. C. Mogul. Emergent (Mis)behavior vs. Complex Software Systems. Technical report HPL-2006-2, HP Laboratories Palo Alto, 2005.

[MWJ+07] G. Mühl, M. Werner, M. A. Jaeger, K. Herrmann, H. Parzyjegla. On the Definitions of Self-Managing and Self-Organizing Systems. In *Proc. of KIVS 2007*. 2007.

[NT04] S. Nakrani, C. Tovey. On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers. *Adaptive Behavior* 12(3-4):223–240, 2004.

[PB04] H. V. D. Parunak, S. Brueckner. Engineering Swarming Systems. In *Methodologies and Software Engineering for Agent Systems*. Pp. 341–376. 2004.

[PBS+08] A. Pokahr, L. Braubach, J. Sudeikat, W. Renz, W. Lamersdorf. Simulation and Implementation of Logistics Systems based on Agent Technology. In *HICL*. 2008.

[RMB+06] U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, H. Schmeck. Towards a generic observer/controller architecture for Organic Computing. In *INFORMATIK*. 2006.

[RS08] W. Renz, J. Sudeikat. Modeling Feedback within MAS: A Systemic Approach to Organizational Dynamics. In *Proc. of OAMAS'08*. 2008.

[RZT07] M. Randles, H. Zhu, A. Taleb-Bendiab. A Formal Approach to the Engineering of Emergence and its Recurrence. In *Proc. of EEDAS 2007*. 2007.

[SGK06] G. D. M. Serugendo, M. P. Gleizes, A. Karageorgos. Self–Organisation and Emergence in MAS: An Overview. In *Informatica*. Volume 30, pp. 45–54. 2006.

[SOR07] H. Seebach, F. Ortmeier, W. Reif. Design and construction of organic computing systems. *Proc. of CEC 2007*, pp. 4215–4221, Sept. 2007.

[SPK06] L. M. Seiter, D. W. Palmer, M. Kirschenbaum. An aspect-oriented approach for modeling self-organizing emergent structures. In *Proc. of SELMAS '06*. 2006.

[SR07a] J. Sudeikat, W. Renz. On Expressing and Validating Requirements for the Adaptivity of Self–Organizing Multi–Agent Systems. *System and Information Sciences Notes* 2(1):14–19, 2007.

[SR07b] J. Sudeikat, W. Renz. Toward Systemic MAS Development: Enforcing Decentralized Self–Organization by Composition and Refinement of Archetype Dynamics. In *Proc. of EEMMAS 2007*. LNAI 5049. 2007.

[SR08a] J. Sudeikat, W. Renz. *Applications of Complex Adaptive Systems*. Chapter Building Complex Adaptive Systems: On Engineering Self–Organizing Multi–Agent Systems, pp. 229–256. IGI Global, 2008.

[SR08b] J. Sudeikat, W. Renz. On the Encapsulation and Reuse of Decentralized Coordination Mechanisms: A Layered Architecture and Design Implications. In *Communications of SIWN*. Volume 4(ISSN 1757-4439), pp. 140–146. 2008.

[SR09a] J. Sudeikat, W. Renz. MASDynamics: Toward Systemic Modeling of Decentralized Agent Coordination. In *Proc. of KIVS 2009*. 2009.

[SR09b] J. Sudeikat, W. Renz. A Systemic Approach to the Validation of Self–Organizing Dynamics within MAS. In *Proc. of AOSE'08*. 2009.

[SRR06] A. Shabtay, Z. Rabinovich, J. S. Rosenschein. Behaviosites: a novel paradigm for affecting Distributed Behavior. In *Proceedings of ESOA'06*. Pp. 23–39. 2006.

[Ste00] J. D. Sterman. *Business Dynamics - Systems Thinking and Modeling for a Complex World*. McGraw–Hill, 2000.

[Zad63] L. A. Zadeh. On the definition of adaptivity. *Proceedings of the IEEE* 51(3):469 – 470, 1963.