Electronic Communications of the EASST
Volume 37 (2011)

Workshops der wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2011 (WowKiVS 2011)

Methods and Tools for Engineering Self-Organizing Software Systems

Ante Vilenica and Jan Sudeikat

12 pages

Guest Editors: Horst Hellbrück, Norbert Luttenberger, Volker Turau
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Methods and Tools for Engineering Self-Organizing Software Systems

Ante Vilenica[1] **and Jan Sudeikat**[2]

Distributed Systems and Information Systems
Computer Science Department, University of Hamburg
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany[1]
Multimedia Systems Laboratory
Hamburg University of Applied Sciences
Berliner Tor 7, 20099 Hamburg, Germany[2]

Abstract: Developing software systems that can cope with constantly changing contexts *automatically* and *at runtime* is a challenging task. Most importantly, changing contexts require *adaptive system behavior* in order to maintain system functionality even under dynamically changing environment conditions. And this relates not only to the core system functionality but also to non-functional aspects such as, e.g., reliability and/or scalability. In order to deal with such challenges adequately, this paper presents an approach that adapts the paradigm of *Self-Organization* to computer software systems. It enables systems to preserve their main caracteristics while changing their structure in consequence to outside influences without the existence of any central controller. So, the specific aim of this paper is to address the utilization of Self-Organization in *engineering* software systems. Accordingly, it presents a set of methods and tools as developed in a recent research project for engineering self-organizing software systems. The underlying approach is organised around state-of-the-art software engineering phases such as system modelling, programming, and the respective software development procedure.

Keywords: Self-Organization, Distributed Systems, Decentralized Coordination

1 Introduction

The development and configuration of (distributed) computer systems is more and more challenged by the rising complexity of these systems. Thereby, complexity of these systems is disclosed by the heterogeneity of software and hardware components, dynamically changing communication partners and connections as well as the spatial distribution of cooperation partners. Even more, complexity is increased through the usage of independent and autonomous components as well as dependencies among these components and dynamically changing contexts. In the end this leads to a situation where the type and availability of system components and resources is constantly changing. In order to enable a mostly autonomous management of computer systems even in such a dynamic environment *adaptive* system characteristics are required. Adaptation [Zad63] means thereby the ability to adjust respectively to customize mostly autonomously the configuration of a computer system in spite of a highly dynamic and unpredictable environment. Currently, there are various research projects that aim at proposing

solutions for the development of adaptive software systems. Prominent examples are the *Autonomic Computing* [KC03] initiative of IBM as well as the priority program *Organic Computing* [BMM+06] of the German Research Foundation. Although these projects have different backgrounds and therefore propose different kinds of blueprints for software architectures they still have one aspect in common. Basically, these projects propose architectures that rely on a hierarchical and centralized design. One advantage of such an approach is the exploitation of common development techniques and design patterns which eases the usage respectively adoption of the approaches for software developers. Examples are the usage of top-down design as well as some kind of central *controller* to manage important decisions, i.e. decisions related to adaptivity, of a software system at runtime. Both of them are quite common and do therefore usually not require additional effort to be learned and applied. At the same time these mentioned blueprints for software architectures have drawbacks that are related to common *non functional requirements* such as scalability, failure tolerance and robustness. From this perspective on, this paper argues that *decentralized* approaches are a valuable path to cope with the challenges related to non functional requirements. Moreover, it is advocated that the concept of *Self-Organization* (SO) is a promising approach to develop software systems that have a completely decentralized architecture and that are capable of managing and changing their structure according to internal and external influences at runtime without external control. Self-organizing processes are characterized by local strategies and local interactions, i.e. design and policies on the micro level, that lead to stable and adaptive structures on the global level. Also, Self-Organization incorporates different *decentral coordination mechanisms* which serve as patterns for the local interaction among components on the micro level.

This work describes the intermediate results of the research project "Selbst-Organisation durch Dezentrale Koordination in Verteilten Systemen"[1](SodekoVS) which is funded by the "Deutsche Forschungsgemeinschaft"[2]. The message of this paper is that is it possible to *engineer* self-organizing software systems. Therefore, this paper gives an outline on methods and tools that where developed in order to be able to engineer the utilization of the paradigm of Self-Organization within software systems. Rather than explaining all tools and methods in detail it aims at giving a guideline how to develop such systems in general. More specific, this guideline tackles aspects that relate to the *development process, modeling and programming* of a software application. It consists of an approach that allows developing functional (local) components independently of the interaction strategy, i.e. decentral coordination mechanism. The aspect of coordination, which is identified as a key aspect, is encapsulated within a layer and a publish/perceive interface and allows for transparent exchangeability of coordination mechanisms at runtime.Also, this separation allows reusing coordination mechanisms in a convenient way. In practice the presented results can be used to solve different problems in very distinct application domains, e.g. for the management of resources in logistics or for clustering in data mining.

The paper is structured as follows: the next section introduces the concept of Self-Organization briefly. Section 3 presents methods and tools that were developed in order to enable software developers to engineer self-organizing software systems. Finally, section 4 gives a conclusion and points out remaining challenges.

[1] Self-Organization Based on Decentralized Coordination in Distributed Systems
[2] German Research Foundation

2 The Concept of Self-Organization

The concept of Self-Organization is quite established and wide spread in natural complex systems. Instances can be found in biology, physics, and chemistry but also in social systems. Well known-examples are molecular self-assembly, spontaneous magnetization and *ant colony optimization* [DS04] which can be used to deal with the traveling salesman problem. Although, there is a plethora of definitions that try to identify the core aspects of Self-Organization there is one definition which tries to point out the most important aspects without having the focus only on a certain domain. *DeWolf* and *Holvoet* propose following definition: "Self-organisation is a dynamical and adaptive process where systems acquire and maintain structure themselves, without external control" [DH04, p.7]. This definition underlines important characteristics of self-organizing systems that reveal why this concept is of special interest for the development of complex distributed systems. It points out the strengths of decentral approaches that are able to deal meaningful with perturbations in the absence of a central controller. This is mostly achieved by systems that consist of many but simple components with local strategies that lead to collective behaviour. In addition, [SGK06] emphasize the ability of self-organizing systems to deal with perturbations at runtime, i.e. to re-organize the structure in order achieve and maintain goals on the macro level.

Closely related to the concept of Self-Organization and sometimes also mixed up is the notion of *emergence*. This concept denotes the occurrence of novel properties, structures or patterns within a system that "are novel w.r.t. the individual parts of the systems" [DH04, p.3]. Therefore, emergence shares similar aspects with Self-Organization as both concepts describe dynamic processes that arise over time and that are robust. However, they are different and can exist isolated in systems.

By analyzing the aforementioned properties of self-organizing processes it is obvious that this paradigm provides characteristics that are desirable for the adaptive and autonomous management of computer systems in a dynamic and unpredictable environment. The applicability of this approach, i.e. the utilization of self-organizing processes in computer systems, has for example been demonstrated in [MMTZ06]. The authors have identified and categorized different self-organizing mechanisms and present case studies where these mechanisms are applied in computer systems. Some examples are: *foraging and brood sorting* can be used for the reallocation of resources in Grid frameworks, *morphogenesis* is useful for the development of artificial immune systems that are used to protect computer systems from malicious code, *molding* can be useful for the self-assembly in robotics and *flocking* for the motion coordination of unmanned airspace vehicles. These examples show the broad variety of self-organizing mechanisms and serve furthermore as a proof of concept for the incorporation of Self-Organization in computer systems. However, an analysis of theses case studies reveals existing challenges. The main problem is that there is a lack of an approach that guides systematically the engineering of self-organizing processes in software applications. Most of existing self-organizing processes are designed *ad hoc* and tailored towards a specific mechanism and application domain [DHS06]. It requires therefore much effort to *reuse* self-organizing processes as they are often intertwined with the application logic and difficult to separate.

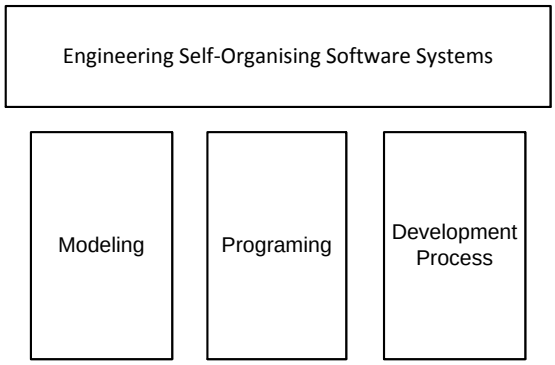

Figure 1: Pillars of the proposed approach

3 Engineering Self-Organizing Software Systems

This section presents the SodekoVS approach for systematically engineering self-organizing processes for software systems. It allows developing these systems purposefully and equipping them with aspects from software engineering like re-usability, transparency and exchangeability. From an abstract point of view it follows the idea of [GC92] that have proposed following equation: Programming = Computation + Coordination. In accordance with this, the SodekoVS approach proposes a *generic system architecture* that allows equipping software systems with self-organizing processes independently of the applied decentral coordination mechanism. Moreover, the latter aspect is used as an interchangeable and transparent glue to manage the components of the system. Thereby, the components contain their core functionality which can be developed isolated from the self-organizing process. Therefore, the generic system architecture gives a solution to the question how to *integrate* self-organizing processes into software systems in a convenient and appropriate way. For clarification, we will call aspects related to this problem as challenges of the *integration level*. In addition to the aspect of programming self-organizing systems the SodekoVS approach also proposes tools and methods that aim at supporting the modeling of Self-Organization as well as the development process. For clarification, we will call this the *level of Self-Organization* as it targets aspects directly related to the utilization of this concept.

In conclusion it can be stated that the approach consists of three pillars (cf. figure 1), i.e. modeling, programming and development process, which target both the level of Self-Organization itself and the level of its integration into software systems. In combination this leads to an approach that guides the engineering of self-organizing processes in software systems. The following subsections will explain the approach in more detail.

3.1 Modeling

This subsection describes a modeling approach that is specificly tailored to the characteristics of self-organizing processes for software systems. According to [HWM06] these processes are characterized by three properties. First, they *form structures*, i.e. specific parts of the application are aligned. In doing so, the type of structure is initially formed but is also able to react and

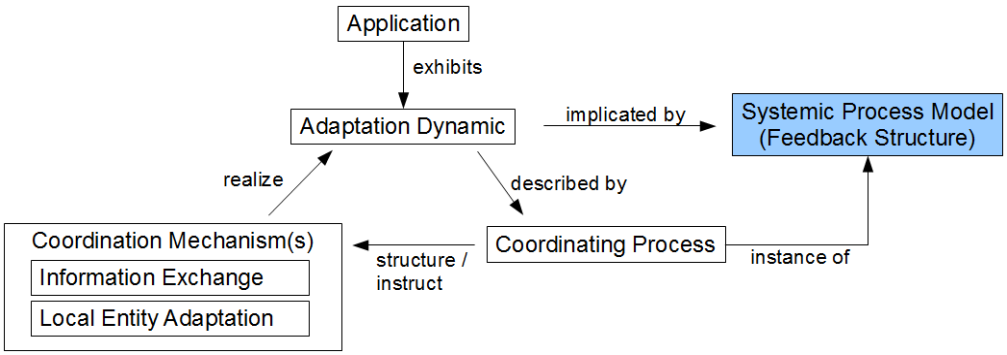

Figure 2: Conceptual model of a decentral coordination process for Self-Organization, following [Sud10]

change to internal and external influences. This reveals the second property of self-organizing processes, in particular *adaptivity*. The third property states that these processes evolve generally through interaction among equal components of the system, i.e. they are *decentral*. All together these properties challenge the modeling of self-organizing processes by applying conventional techniques from software engineering. Therefore, a new approach has been proposed that tackles these challenges. Figure 2 depicts the idea of this approach. Decentral *Coordinating Processes* are identified as independent elements of the application design and implementation. Applications exhibit certain dynamics and thereby those dynamics are of special interest that lead to the ability to adapt to changes in the context of the application. This behavior, i.e. adaptivity, of the software system can be described and enforced at runtime as a coordination process that is part of the application and that manages the behavior of the system. In order to realize these coordination processes in software systems *coordination mechanisms* are required. They can be categorized into two groups [Sud10]: *Information Exchange* and *Local Entity Adaption*. The former are in charge of spreading out information in the system and w.r.t. to Self-Organization mechanisms as *pheromones* [BC06] or *computational fields* [MZ05] have to be mentioned. In contrast, the latter ones are in charge of enabling system components to change the local behavior, i.e. mechanisms as control systems [Bro86] or stochastically role based approaches [WSHG04]. The combination of both types of coordination mechanisms allows establishing *interdependencies* among components of the system as information can be distributed and evaluated by single entities and cause eventually changes of local behavior. From a more formal point of view, the interdependencies among system components can be formalized in a systemic process model. This model describes the general structure of the coordination process as a network that contains the system components and their interdependencies. For a concrete application this general structure is instantiated and refined with parameters for the contained mechanisms. The parameterization influences the dynamic behavior of the system at runtime. Hence, systemic process models define *classes of interdependency* structures that can be used to create various concrete coordination processes.

It has been shown that the phenomena that self-organizing systems exhibit can be explained with (distributed) feedback loops [BDT99, BMG+09]. Using feedback loops to model and analyze the interdependency structure of a software system enables engineering the dynamic behavior of the system on a global level. With this modeling approach it is possible to detect undesired

phenomena as oscillations and fixed points. Moreover, by adding additional feedback loops to the original system model it is possible to prevent undesired phenomena. This modeling approach is different to other approaches as [Edm04, Ger07] since it does not model the desired phenomena, i.e. system structures, itself. In contrast, it models and adjusts the reasons, i.e. feedback loops, which cause the structures. It is therefore a bottom-up modeling approach. More specific, the proposed modeling approach bases on established techniques from the research area of system dynamics and tailors them towards the specific requirements of agent-based distributed systems. On the one hand it utilizes *causal loop diagrams* [Ste00] which are used to graphically model the variables of a system and their interdependencies. On the other hand it uses an *agent causal behavior graph* (ACBG) [Sud10] as a more formal representation of the elements, i.e. software agents, and their interdependencies.

Additionally, the developed systemic modeling approach has been used to describe and classify well-known self-organizing processes. These processes are catalogued according to the properties of the structure as well as the characteristics of the structure they evolve. This classification extends existing catalogues [MMTZ06, DH06] as it adds two new properties to describe self-organizing processes for software systems. The first property describes the structure of the interdependencies among system components as they cause feedback loops which lead to adaptive behavior on system level. Another property targets the description of the structure that these loops cause on system level. Thereby, it has to be pointed out that the catalogue characterizes self-organizing processes *independently* of a specific application domain. Moreover, it describes the characteristics of the structure and evolved phenomena in general and can therefore be used to select the appropriate process for a specific software system. More details about the catalogue can be found in [Sud10]. In conclusion it can be stated that the developed method and tool support a systematic modeling of self-organizing processes for software systems. It supports the re-usability of theses processes as their structure and dynamics can be modeled and analyzed independently of a specific application.

3.2 Programming

The goal of this section is to present an approach that enables the programming of self-organizing systems. It targets therefore the level of integration as it shows how application developers can equip (existing) software systems with self-organizing properties. Figure 3 depicts the blueprint of an architecture that guides the integration. It is a layered architecture which has the functionality of the software system as it is perceived by users at the top layer. Hence, it is assumed that the software system is realized by a multi agent system, i.e. a system that consists of (many) single components (so called agents) which offer certain functions and act thereby autonomously and proactively. By interacting with each other the single agents are able to provide functionality at system level. Agents are executed on a distributed middleware platform which offers general functionalities that are required for an execution of a multi agent system. Between the bottom layer, i.e. the execution infrastructure, and the top layer the architecture proposes a coordination layer which encapsulates all aspects related to the task of coordination. On the one hand the coordination layer consists of *coordination media* which serve as an infrastructure for communication by offering wide-spread interaction mechanisms via a generic publish/subscribe interfaces. On the other hand it contains *coordination endpoints* which serve as mediators be-

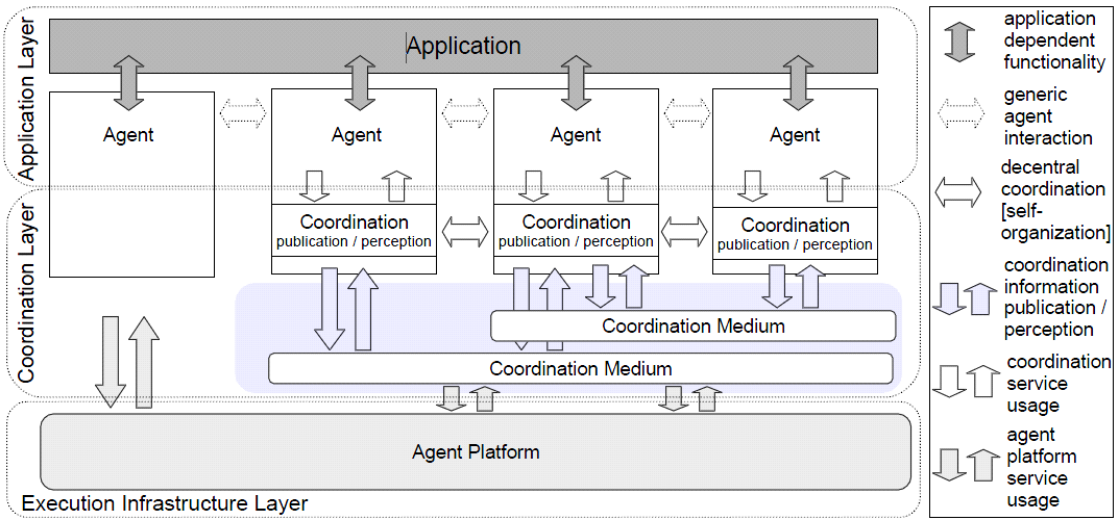

Figure 3: Blueprint of an architecture for self-organizing software systems, following [Sud10]

tween the local components, i.e. agents, and the coordination media. Endpoints are software components that observe the execution of agents and are also able to modify them. Thereby, endpoints encapsulate the coordination logic and perform automatically the tasks which are related to the successful execution of coordination. These tasks contain all aspects related to the interaction with other components, i.e. to initiate interaction as well as to participate. This happens w.r.t. to observations of the agent as well as the coordination media. Moreover, the endpoint has the possibility to modify the state of the agent if it is required.

The proposed architecture extends related approaches that aim at integrating decentral coordination of agents at two aspects. First, it emphasizes a clear separation of application logic and coordination logic. Second, the blueprint allows integrating coordination in an almost non-invasive manner, i.e. it is possible to execute automatically activities which are required for the coordination among components of the system.

To proof the concept of the architecture two execution platforms have been implemented. They differ in the way they realize the coordination layer. One implementation ([Sud10]) is inspired by the paradigm of distributed event-based systems and proposes a distributed approach for the realization of coordination endpoints. The other implementation ([VSL+10]) realizes the endpoint functionality as a functionality of the environment where the agents are situated. It introduces the concept of *coordination spaces* as a dedicated part of environments which are in charge of encapsulating aspects related to the task of agent coordination. Both implementations are realized on the *Jadex Agent Framework*[3].

In addition to the reference architecture and its implementations the programming of self-organizing software systems is supported by the XML-based configuration language *MASDynamic* [Sud10]. This language allows specifying details of ACBG-based process definitions in order to enable their automatic execution within the reference implementation. MASDynamic acts therefore as a connector between an abstract process definition and the implementation of

[3] http://jadex.informatik.uni-hamburg.de

the system components that are coordinated. It contains two description levels: the abstract representation of generic process structures that are independent from a concrete software system and the detailed mapping of process elements to components in the realization of an agent-based software system. More specific, MASDynamic allows specifying exactly the structure of interactions among components. First, it describes the kind of component behavior which is of interest for the self-organizing process. Second, it determines which coordination medium to use. Third, it specifies which component behavior can be influenced by interactions. Hereby, component behavior is specified w.r.t. to elements of agents as goals, plans, beliefs etc. MASDynamic itself is designed as a general language which abstracts from specific interaction techniques. Additionally, MASDynamic can be used as a basis to define classes of coordination processes which again can be further refined and parameterized for specific applications.

3.3 Development Procedure

The conception of self-organizing problem solving strategies requires considerate expertise. For software development teams this is particularly challenging since self-organizing phenomena are interdisciplinary objects of research. Within the SodekoVS project, this is reflected by the adoption of a modelling approach (see s ection 3.1) for the description of complex adaptive systems [Ste00]. The programming model continues the modelling approach as it enables to transform systemic models of processes to detailed prescriptions of process instances that can be executed by the proposed middlware layer.

Due to the inherent challenges and the adoption of a tool set that reflects the interdisciplinary nature of self-organizing systems, guidance in the application development is needed. Therefore, a detailed development procedure has been revised. The conception of a process that creates, due to bottom-up causation, the intended system level structures is non-trivial and creative act. The procedure can not resolve these issues but the fundamental development strategies, activities and detailed steps have been identified to guide and plan for the application development [Sud10].

Conceptually, this procedure is conceived as an extension to established development processes. An early evaluation of development methodologies for agent based systems has shown that development processes and modelling approaches are typically biased towards specific development platforms [SBPL04]. Consequently, development teams benefit from using the right match of an (implementation) platforms and methodological support. When addressing self-organizing solutions the selection of the implementation tools and the related methodology should be affected as less as possible. Therefore, the conception, refinement and integration of a self-organizing dynamic is understood as an supplement to the conventional application development. Method engineering techniques for MAS [CGGS07], in particular the *Software & Systems Process Engineering Metamodel Specification* (SPEM)[4] is used to describe the process extension in a standardized format. Using this approach, the integration of self-organizing properties can be addressed by development teams when needed. For example, experimentation with prototypes of elaboration of the system requirements can reveal the need for adjustments of the dynamical system behaviour by decentralized coordinating processes.

The extension to development processes is illustrated in figure 4. It shows a set of development

[4] http://www.omg.org/spec/SPEM/2.0/

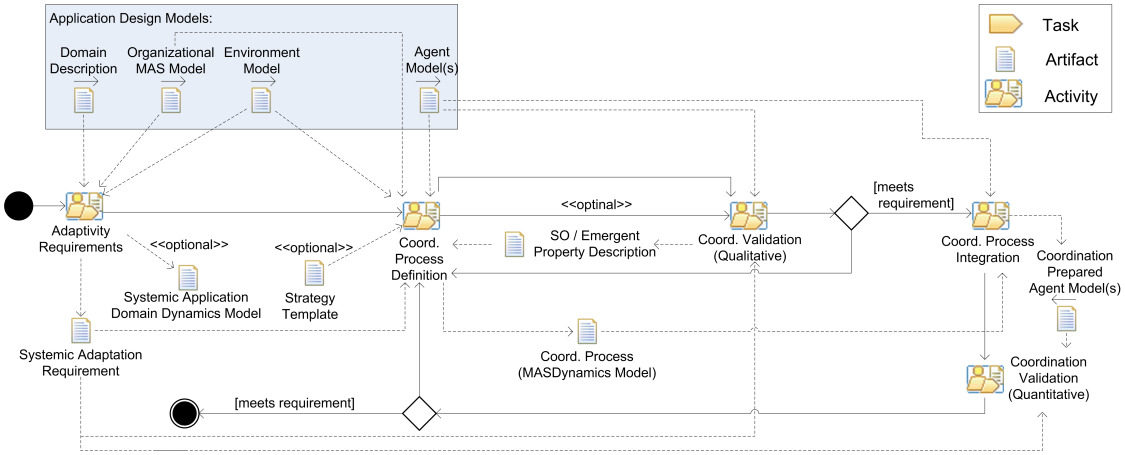

Figure 4: The development and integration of a self-organization process in an agent-based software system [Sud10]

activities and development artefacts that are conceptually related to the self-organizing aspects of the MAS. Following the SPEM conceptual model, development activities describe reusable aspects of the application development and artefacts refer to the elements that are created and/or modified by these activities. The denoted activities supplement the fundamental Requirements, Analysis, Design, and Testing disciplines in software development (e.g. see [JBR99]). The activities are interlinked with the application development via a set of *Application Design Models*. These originate in the conventional application development and describe the agent-based software system. When these models change during incremental development, developers check whether the SO-related activities have to be repeated or SO-specific artefacts have to be updated. The *Adaptivity Requirements* activity addresses the description of the intended system behaviour [SR07]. The resulting model (*Systemic Adaptation Requirement*) describes how the system is expected to adapt at run-time. Based on the intended behaviour an appropriate process model is derived in the *Coordinating Process Definition* activity [SR10]. A supplement is factored out by comparing the models of the intended dynamics with the current application behaviour. When properly integrated, this addition is capable to enhance the operation of the software system. Before the actual implementation, the abstract model of the processes can be validated via system simulations (*Coordination Validation (Qualitative)*) [SRRT09]. After it is validated that the process is in principle capable to bring about the desired dynamics, it is integrated in an actual system (*Coordinating Process Integration*) [SR09]. This is based on the mapping of the process elements to the elements in the system implementation and annotating the information that are needed to enable the automated execution, using the corresponding execution middleware. The outputs of this activity are the Process model (*Coordinating Process*) and the software agents that are prepared to participate in this process (*Coordination Prepared Agent Model(s)*). Finally, it is validated that the supplementation of the inter-agent process leads to the intended system level effects. The detailed descriptions of the contained activities in SPEM notation can be found in [Sud10].

4 Conclusion & Remaining Challenges

This work has motivated that current tendencies in computer systems require software applications to be able to adapt to (unpredictable) changes at runtime. The complexity of these systems demands for solutions that are able to manage the adaptive behavior automatically, i.e. without manual effort. Inspired by the wide-spread and successful paradigm of Self-Organization in natural systems it is advocated that this paradigm is also suitable for the development of adaptive computer systems. Moreover, due to the inherent distributed architecture self-organizing systems are an adequate approach for providing systems with non-functional requirements as scalability and failure tolerance. In order to benefit from the advantages of Self-Organization and to cope with challenges this work has presented a systematic approach for developing software systems that exhibit self-organizing behavior. This approach guides the engineering of these systems by presenting methods and tools that target the modeling, programming and development process of these systems. Therefore, the present approach shows that it is possible to apply the paradigm of Self-Organization to computer systems in general.

Future work will strive on the one hand towards further improving the usability of the approach. At the current stage the usage of the methods and tools requires (considerable) learning effort for application developers which are not familiar with the domains of system dynamics or SPEM. It is envisioned to provide more high-level interfaces that hide the complexity and ease therefore the utilization in software projects. As part of this goal, it is also envisioned to provide *best practices*. On the other hand future work will target the aspect of enabling and supporting the exchange of self-organizing processes at runtime. This might be necessary for systems which contain two or more different modes of operation, i.e. different goals on system level require different self-organizing processes. Thereby, goals of systems can change at runtime due to events in the environment. This challenge requires the development of appropriate modeling techniques as well as programming tools.

Acknowledgements: The authors would like to thank Deutsche Forschungsgemeinschaft (DFG) for supporting this work through a research project on ”Self-organization based on decentralized co-ordination in distributed systems” (SodekoVS). Furthermore, we would like to thank Winfried Lamersdorf, Wolfgang Renz, Lars Braubach and Alexander Pokahr for continued inspiring discussion and fruitful joint work.

Bibliography

[BC06] S. Brueckner, H. Czap. Organization, Self-Organization, Autonomy and Emergence: Status and Challenges. *ITSSA* 2(1):1–10, 2006.

[BDT99] E. Bonabeau, M. Dorigo, G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[BMG+09] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, M. Shaw. Software Engineering for Self-Adaptive Systems. In Cheng

et al. (eds.). Chapter Engineering Self-Adaptive Systems through Feedback Loops, pp. 48–70. Springer, 2009.

[BMM+06] J. Branke, M. Mnif, C. Müller-Schloer, H. Prothmann, U. Richter, F. Rochner, H. Schmeck. Organic Computing – Addressing Complexity by Controlled Self-organization. In *Proc. of ISoLA*. IEEE Computer Society Press, 2006.

[Bro86] R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of* 2(1):14 – 23, 1986.

[CGGS07] M. Cossentino, S. Gaglio, A. Garro, V. Seidita. Method fragments for agent design methodologies: from standardisation to research. *Int. J. Agent-Oriented Software Engineering* 1(1):91–121, 2007.

[DH04] T. DeWolf, T. Holvoet. Emergence and self-organisation: a statement of similarities and differences. In *Proceedings of the International Workshop on Engineering Self-Organising Applications*. Pp. 96–110. Springer, 2004.

[DH06] T. De Wolf, T. Holvoet. A catalogue of decentralised coordination mechanisms for designing self-organising emergent applications. Technical report CW458, K.U.Leuven, Department of Computer Science, 2006.

[DHS06] T. De Wolf, T. Holvoet, G. Samaey. Development of Self-organising Emergent Applications with Simulation-Based Numerical Analysis. In Brueckner et al. (eds.), *Engineering Self-Organising Systems*. LNCS 3910, pp. 138–152. Springer, 2006.

[DS04] M. Dorigo, T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[Edm04] B. Edmonds. Using the Experimental Method to Produce Reliable Self-Organised Systems. In Brueckner (ed.), *Engineering Self-Organising Systems: Methodologies and Applications*. LNAI 3464, pp. 84–99. Springer, 2004.

[GC92] D. Gelernter, N. Carriero. Coordination languages and their significance. *Commun. ACM* 35:97–107, 1992.

[Ger07] C. Gershenson. *Design and Control of Self-Organizing Systems*. PhD thesis, Vrije Universiteit Brussel, 2007.

[HWM06] K. Herrmann, M. Werner, G. Mühl. A Methodology for Classifying Self-Organizing Software Systems. *ITSSA* 2(1):41–50, 2006.

[JBR99] I. Jacobson, G. Booch, J. Rumbaugh. *The unified software development process*. Object Technology Series. Addison Wesley, 1999.

[KC03] J. O. Kephart, D. M. Chess. The Vision of Autonomic Computing. *Computer* 36(1):41–50, 2003.

[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.

[MZ05] M. Mamei, F. Zambonelli. *Field-Based Coordination for Pervasive Multiagent Systems (Springer Series on Agent Technology)*. Springer, 2005.

[SBPL04] J. Sudeikat, L. Braubach, A. Pokahr, W. Lamersdorf. Evaluation of Agent–Oriented Software Methodologies - Examination of the Gap Between Modeling and Platform. In *Agent-Oriented Soft. Eng. V, 5. Int. Work. AOSE 2004*. Pp. 126–141. 2004.

[SGK06] G. D. M. Serugendo, M. P. Gleizes, A. Karageorgos. Self-Organisation and Emergence in MAS: An Overview. *Informatica (Slovenia)* 30(1):45–54, 2006.

[SR07] J. Sudeikat, W. Renz. On Expressing and Validating Requirements for the Adaptivity of Self-Organizing Multi-Agent Systems. *Syst.and Inf. Sc. Notes* 2(1):14–19, 2007.

[SR09] J. Sudeikat, W. Renz. Programming Adaptivity by Complementing Agent Function with Agent Coordination: A Systemic Programming Model and Development Methodology Integration. *Com. of SIWN* 7:91–102, 2009.

[SR10] J. Sudeikat, W. Renz. On the Modeling, Refinement and Integration of Decentralized Agent Coordination – A Case Study on Dissemination Processes in Networks. In *Self-Organizing Architectures*. LNCS 6090, pp. 251–274. Springer, 2010.

[SRRT09] J. Sudeikat, M. Randles, W. Renz, A. Taleb-Bendiab. A Hybrid Modeling Approach for Self-Organizing Systems Development. *Com. of SIWN* 7:127–134, 2009.

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

[Sud10] J. Sudeikat. *Engineering Self-Organizing Dynamics in Distributed Systems: A Systemic Approach*. PhD thesis, University of Hamburg, Dept. of Informatics, 2010. http://www.sub.uni-hamburg.de/opus/volltexte/2010/4928/

[VSL+10] A. Vilenica, J. Sudeikat, W. Lamersdorf, W. Renz, L. Braubach, A. Pokahr. Coordination in Multi-Agent Systems: A Declarative Approach using Coordination Spaces. In Trappl (ed.), *Proc. of the 20th EMCSR - Int. Work. From Agent Theory to Agent Implementation*. Pp. 441–446. Austrian Soc. for Cybernetic Studies, 4 2010.

[WSHG04] D. Weyns, K. Schelfthout, T. Holvoet, O. Glorieux. A role based model for adaptive agents. In *Proceedings of the AISB 2004, Fourth Symposium on Adaptive Agents and Multi-Agent Systems*. Pp. pp. 75–86. 2004.

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