Observation and Control of Organic Systems

Sven Tomforde[1], Holger Prothmann[2], Jürgen Branke[3], Jörg Hähner[1], Moez Mnif[1], Christian Müller-Schloer[1], Urban Richter[2], and Hartmut Schmeck[2]

[1] Leibniz Universität Hannover, Institute of Systems Engineering, Appelstr. 4, 30167 Hannover, Germany,
{tomforde|haehner|mnif|cms}@sra.uni-hannover.de
[2] Karlsruhe Institute of Technology (KIT), Institute AIFB – Bldg. 05.20, 76128 Karlsruhe, Germany,
{holger.prothmann|urban.richter|hartmut.schmeck}@kit.edu
[3] University of Warwick, Warwick Business School, Coventry, CV4 7AL, UK,
juergen.branke@wbs.ac.uk

Summary. Organic Computing (OC) assumes that current trends and recent developments in computing, like growing interconnectedness and increasing computational power, pose new challenges to designers and users. In order to tackle the upcoming demands, OC has the vision to make systems more life-like (organic) by endowing them with abilities such as self-organisation, self-configuration, self-repair, or adaptation. Distributing computational intelligence by introducing concepts like self-organisation relieves the designer from exactly specifying the low-level system behaviour in all possible situations. In addition, the user has the possibility to define a few high-level goals, rather than having to manipulate many low-level parameters.

This article presents the generic Observer/Controller design pattern that serves as an architectural blueprint for self-organised OC systems. The major components of the Observer/Controller architecture and their responsibilities are introduced. Besides the general design, we discuss several distribution variants of the architecture. Furthermore, a survey presents research projects that use the Observer/Controller paradigm to solve technical problems from various domains.

1 Introduction

During the last decades, an impressive progress in computing technology has led to an exponential increase in available computing power and a decrease of chip size to a miniature format. The combination of both trends allows for an ubiquitous presence of small and interconnected devices that simplify today's life. The advantages of these new achievements are manifold as systems become more flexible and multifunctional and thereby almost indispensable in daily life. But these benefits are accompanied by several drawbacks. Some years ago, malfunctions of technical devices due to mutual influences, unpredictable environmental conditions, or decreasing manageability caused by vast

configuration spaces have not been considered as a practical problem. In order to cope with these aspects of complexity and the corresponding challenges in system design, Organic Computing (OC) proposes to distribute the computational intelligence among large populations of smaller entities and to provide appropriate degrees of freedom for self-organised behaviour [21].

The most important aspect of OC systems in this context is the adaptability to changes in the environmental conditions, in particular with respect to human needs. Hence, an "organic" system should be aware of its own capabilities – typically referred to as *self-x properties*. Originally defined by the Autonomic Computing community [9] in the context of self-managed IT-infrastructure, these properties include characteristics like self-configuration, self-healing, self-optimisation, and self-protection. Freedom in terms of self-organisation and self-x properties provides the basis for the anticipated adaptiveness and allows for reducing the complexity of system management.

The new requirements are supported by a paradigm shift in system design – dynamic, adaptive, and self-managing systems are needed instead of monolithic and static systems [15]. This has impact on the work of a system designer, because the existing techniques and models cannot be reused. Thus, the question arises, how such self-organised and highly interconnected OC systems can be designed, and how they can be made reliable and usable. Obviously, the designer is neither able to foresee all possible system configurations, nor to prescribe proper behaviours for all cases. In addition, we need to relieve the user from controlling all system parameters in detail – in contrast, we need mechanisms that allow to influence the system on a higher level by defining strategies or setting goals.

The paradigm shift in system development is accompanied by the need of adequate methods, techniques, and system architectures to be able to design and control complex systems, since there is no existing generalised approach. Therefore, a regulatory feedback mechanism – the *generic Observer/Controller architecture* – has been proposed [3, 20]. The architecture constitutes a generalised way to achieve controlled self-organisation in technical systems. This article summarises this generic Observer/Controller design pattern. Besides an explanation of the concept and its main components, different design variants are introduced. Furthermore, the article gives an overview of applications that are built upon the presented design pattern or are at least inspired by it.

This article is organised as follows: Section 2 describes the Observer/Controller design paradigm for OC systems. This design paradigm formulates a generic approach which has to be customised to the specific application. Section 3 describes possible design variants and outlines their particular advantages and disadvantages. Although the architecture is a recent theoretic concept, there are already some systems that are based on the general design. Some of these systems are classified and outlined in Sect. 4. Finally, Sect. 5 summarises the presented design and outlines further application domains and extensions of the proposed architectural concept.

2 Generic Observer/Controller Architecture

OC systems are characterised by the need of an adequate response to environmental or internal changes. Typically, this response results in an adaptive behaviour and incorporates further aspects like robustness and flexibility. In order to allow for such an adaptation process, the system's design has to provide a regulatory feedback mechanism that is capable of monitoring, analysing, and reacting to changing conditions. Therefore, we propose a generic Observer/Controller architecture, which constitutes a generalised way to achieve controlled self-organisation in technical systems [3, 20]. As depicted in Fig. 1, this regulatory feedback mechanism contains three major components:

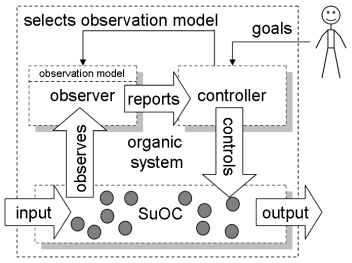


Fig. 1. Observer/Controller architecture

System under Observation and Control (SuOC) The SuOC is the "productive" part of the system that serves a specific purpose. It is functional without Observer and Controller and it will remain operable if higher layers fail (i.e. Observer/Controller components).

Observer The SuOC's state and dynamics are monitored by the Observer in order to give an appropriate description of the current situation for the whole system at each point of time.

Controller Based on the Observer's aggregated information, the Controller influences the SuOC with respect to the goals given by the user.

Figure 2 illustrates the architectural design in more detail and describes the workflow within the whole system. The regulatory feedback mechanism of the generic Observer/Controller architecture has some similarities with other architectural concepts like IBM's MAPE cycle [9]. Since this article does not discuss related architectural concepts, the reader is referred to *Richter* [19] for a comparative survey in the context of the generic Observer/Controller architecture. The remainder of this section focuses on describing the depicted key components of the architecture and their tasks.

2.1 System under Observation and Control

The lowest layer of the architecture encapsulates the productive part of the system. This productive system can serve various purposes, examples range from urban traffic light control [17] over elevator control [18] to data communication systems [24] (see Sect. 4 for a survey). Higher layers of the architecture monitor and adjust (if necessary) the parameter configurations of the productive system in discrete time intervals.

OC postulates the distribution of computational intelligence among large populations of smaller entities – thus, the SuOC in Fig. 2 might refer to single

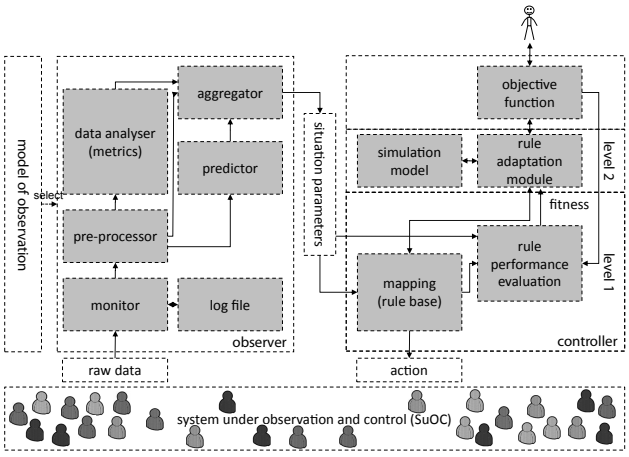


Fig. 2. Generic Observer/Controller architecture

systems or groups of autonomous systems. In both cases, the SuOC needs to fulfil some basic, application-specific requirements:

- The SuOC's behaviour and its environmental conditions have to be observable.
- The performance of the SuOC according to some goal given by the designer or user has to be measurable.
- The SuOC has to possess a set of variable parameters that can be dynamically adapted at run-time and that have certain impact on the performance of the system.

2.2 Observer

It is the observer's task to measure and quantify the current state of the SuOC and to predict its future development. The observation process consists of the steps monitoring, pre-processing, data analysis, prediction, and aggregation. An observation model customises the observer by selecting the observable attributes of the SuOC and by determining appropriate analysis and prediction methods.

Monitor and log file The monitor samples attributes of the SuOC according to a given sampling rate. All measured data is stored in a log file for every loop of observing/controlling the SuOC. The stored time series form the basis for the pre-processing, analysis, and prediction steps.

Pre-processor Typical tasks performed during pre-processing include the smoothing and filtering of stored time series and the extraction of derived attributes. The pre-processed data is used in the data analysis and prediction steps.

Data analyser The data analyser provides a system-wide description of the SuOC's current state. The implemented analysis techniques largely depend on the observed system and the purpose of the Observer/Controller. Cluster computation, emergence detection (according to the definition in [14]), or further mathematical and statistical methods can be applied to extract information from the pre-processed data.

Predictor While the data analyser is dedicated to the current system state, the predictor's task is to forecast future developments. This enables the controller to base its control decisions not only on historic and current data, but also on predicted developments. Prediction techniques are again specific to the organic system's domain.

Aggregator The results of the pre-processor, data analyser, and predictor are handed on to the aggregator where they are combined to situation parameters which are transmitted to the controller.

Depending on the requirements of the controller, the obtained situation parameters can be adapted using the *model of observation*. The model specifies which properties of the SuOC are observed and which sampling rate is used for observation (this selection is obviously limited by the available detectors and their capabilities). Furthermore, the applied analysis and prediction methods are selected. By influencing the model of observation, the controller can focus in detail on critical aspects within the SuOC even when its resources (like communication bandwidth, processing power, or energy) are limited. Further details on the Observer with a special focus on determining emergent behaviour have been discussed by *Mnif* [13].

2.3 Controller

Based on the received situation parameters, the controller influences the SuOC to achieve the objectives specified by the user. The controller is internally composed of two layers: *Layer 1* is dedicated to the on-line learning of appropriate actions for the received situation parameters. It consists of a mapping component that assigns possible actions to known situations. Furthermore, the performance of stored situation-action mappings that have been applied in the SuOC is evaluated by a performance evaluation component. *Layer 2* aims at improving the mapping of Layer 1 by providing off-line optimisation capabilities. It is composed of an adaptation module and a simulation model. Using optimisation or machine learning algorithms, the adaptation module creates additional mappings and can rely on the simulation model in the process. The components of both layers are discussed in the following:

Mapping The mapping component is responsible for an immediate reaction to received situation parameters. It stores previously learnt situation-action mappings (e.g. in form of Learning Classifier System rules [4]) which determine the reaction to known situations. When a situation is unknown, an action might be deduced from stored mappings for similar situations.

Performance evaluation The performance evaluation component calculates quality updates for the situation-action mappings based on their performance observed in the SuOC. When the execution of an action changes the state of the SuOC, this change is subsequently reflected in the situation parameters derived by the observer. The controller's performance evaluation component evaluates the change with respect to the objective function and updates previously applied situation-action mappings through the adaptation module. Machine learning techniques like Reinforcement Learning [22] can be applied here. The update of existing mappings resembles an on-line learning based on observations in the SuOC.

Adaptation module The adaptation module's main task is to create new situation-action mappings and to delete mappings of insufficient quality. For the exploration of additional mappings, various optimisation techniques (including, e.g., Evolutionary Algorithms [7]) can be applied. Optimisations are supplemented by a simulation model that allows for the safe evaluation of candidate solutions. The adaptation module resembles an off-line optimisation that is performed in parallel to the on-line learning mentioned above.

Simulation module The simulation module supports the exploration of new situation-action mappings. It allows to quickly and safely estimate the effect of a new action or mapping before its application in the SuOC.

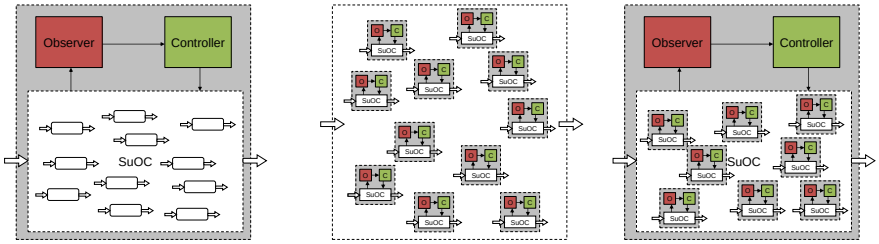
Objective function The user-defined objective function guides the learning and optimisation processes on both layers of the controller. It is used for fitness evaluation in the performance and adaptation modules.

The combination of on-line learning (Layer 1) and off-line optimisation (Layer 2) in the Controller leads to a *2-layered learning* concept that is inspired by the *Anytime Learning* approach that has been proposed by *Grefenstette and Ramsey* [8]. The most prominent examples of 2-layered learning in OC systems are traffic control [17] and data communication networks [24]. A detailed investigation of the Controller with a special focus on learning aspects has been performed by *Richter* [19].

3 Design Variants of the Observer/Controller Architecture

The generic Observer/Contoller architecture needs to be customised to different scenarios by adapting the various components of the Observer and the Controller. As stated in [5] and depicted in Fig. 3, this customisation of design variants ranges from fully central to fully distributed architectures. The former case describes a single Observer/Controller that regulates various components of the SuOC and that directly intervenes into all of these entities (see Fig. 3(a)). In contrast, the latter example defines one Observer/Controller for each component of a technical system (see Fig. 3(b)). These two variants – the

fully central and the fully distributed architecture – define the two extreme points in the design space. Nevertheless, there are also many other distribution possibilities like a multi-level architecture (see Fig. 3(c)).



(a) Central (b) Distributed (c) Multi-level

Fig. 3. Distribution possibilities of the generic Observer/Controller architecture

Based on these various possibilities to realise and customise the generic Observer/Controller architecture, the designer of a technical system has to decide about the most promising approach for his/her context. In the course of this decision process, the need for different design variants can be classified according to increasing size, complexity, and heterogeneity of the contained subsystems. The simplest case is an isolated system with a clearly defined purpose and a restricted configuration space where no distribution is needed. In contrast, larger and more complex systems are characterised by a drastic increase of situation and configuration spaces that cannot be handled by one single Observer/Controller component. With growing complexity, a hierarchical or multi-levelled decomposition of the control problem becomes more recommendable. A common analogy for multi-levelled systems is the organisational structure of large companies: The management serves as highest instance that defines abstract global goals or strategies. Lower layers of the hierarchy convert these abstract goals for their area of responsibility into more specific goals – hence, high level administration units are not involved in low level decisions.

OC introduces the variability of systems as a measurement for the quantification of complexity during the design process for technical systems. The term *variability* is defined as the number of possible configurations of a SuOC [6] (see Cha. 1.1). Obviously, the variability tends to increase with the complexity of the SuOC. However, introducing hierarchical and multi-levelled Observer/Controller structures is a powerful instrument to reduce the externally visible variability and, therefore, hide the complexity of a system.

4 Application Survey

The generic architectural design (as presented in Sect. 2) and its distribution variants (as presented in Sect. 3) describe an architectural blueprint. Several

projects from varying application domains have developed their own systems that are built upon this basic concept or are at least inspired by it. This section provides a survey of various applications. The survey is structured according to the distribution of the Observer/Controller components.

4.1 Central Observer/Controller

When a technical system cannot be subdivided into subsystems or when its subsystems are highly interrelated, a centralised Observer/Controller architecture (like in Fig. 3(a)) is often the method of choice for monitoring and controlling the system's behaviour. Applications that have been successfully implemented with a centralised Observer/Controller include, but are not limited to, elevators, cleaning robots in buildings, and machine management in off-highway machines.

Elevator Control

An *organic elevator control* system [18] serves as one example for a centralised variant of the generic Observer/Controller architecture. In a group of elevators, emergent effects can appear that have to be detected and controlled to avoid undesired processes. Typically, elevators stop at the nearest hall call in their current running direction and only change their direction after serving all requests for the current one. In buildings with several parallel elevators working according to this simple concept, a synchronisation effect can be observed when all elevators move up and down as a parallel wave. This so-called *bunching* effect results in increasing waiting times for passengers and has been proven as inefficient [1]. From the viewpoint of OC, the bunching effect represents an interesting emergent behaviour in a technical scenario which has to be detected and avoided. Therefore, the Observer contains corresponding emergence detectors and the Controller can intervene to discontinue the synchronisation by manipulating the behaviour of individual elevator cabins. As one example, the elevator does not notice calls for the next hall and passes them without stopping. Considering the distribution variants given in Sect. 3, the organic elevator control system consists of several autonomous elevators and one centralised Observer/Controller component.

Organic Computing in Off-highway Machines

The project *Organic Computing in Off-highway Machines* (OCOM) focuses on machine management in off-highway machines like tractors or wheel loaders [28]. An off-highway machine consists of several subsystems (like the traction drive, the power take-off, the hydraulic system, and varying auxiliary components) that are closely interrelated. An adaptive machine-wide management of these subsystems is crucial for the machine's efficient operation (e.g. with respect to a minimal fuel consumption).

To adapt the subsystems to the machine's various operation scenarios and to improve their coordination, OCOM relies on an Observer/Controller architecture that builds upon the generic concept presented in this article. The SuOC is formed by the machines' subsystems, while the Observer/Controller is responsible for a reliable, adaptive, and robust machine management. The major contribution of the project is the modular development and implementation of the generic design pattern.

To support the basic investigations, a simulation model for OC-based off-highway machines has been developed. The model allows to investigate the quantification and control of emergent global effects which can occur in a collection of subsystems. Based on these simulative studies, comprehensive measurements are conducted on a real off-highway machine (a *Fendt Vario 412* tractor) rigged with additional sensor technology. Future work will refine the architectural design and generalise the achieved results to improve the control strategies of similar machines. Further details on OC in off-highway machines are given in Cha. 6.1.8.

Cleaning Robots

Self-organised *cleaning robots* have been investigated as another application scenario for the generic architecture [19]. These cleaning robots follow a local strategy to search in their local neighbourhood for dirty places, clean these places, and self-optimise their search behaviour by learning. The group of robots is applied to a dirty environment and tries to maximise the cleaning performance. Robots are able to indirectly communicate with each other by placing "pheromones" at places they have already cleaned – these pheromones are observed by other robots. The strategy of the robot is based on avoiding double-cleaning of areas. Based on this simple scenario, two different variants of the generic architecture have been investigated: a centralised and a distributed variant. In the former scenario, one centralised component is responsible for generating and exchanging the robots' behaviour strategies [16], while the latter scenario focuses on distributed learning of autonomous robots without the need of centralised control [12].

4.2 Distributed Observer/Controller Components

When a technical system consists of several subsystems, it can be advisable to observe and control each subsystem with a separate Observer/Controller component (see Fig. 3(b)). The predominant application area of distributed OC systems are applications where the subsystems are loosely coupled, where the subsystems are locally distributed (like in communication or road networks), or where subsystems are controlled by different authorities (like in the context of service-oriented architectures).

Service-oriented architectures (SOAs) are typical distributed applications with manifold interactions among various components. Since conventional

management systems are restricted in their capabilities to deal with sophisticated interactions between components, management responsibility at run-time is distributed to each component. To this end, design principles of OC are leveraged to design Organic Service-oriented Architectures (OSOAs) [23]. In detail, each SOA entity is equipped with an Observer/Controller component to achieve controlled self-organisation locally [11]. The Observer monitors its respective SOA component and determines its current operational state. Based on this information, the Controller adapts the behaviour of the underlying SOA component according to given objectives specified in service-level agreements (SLA). In addition, service components coordinate their run-time behaviour via automatically negotiated SLAs, which ensures a compliance of the entire SOA-based system with given business objectives [10]. As result, the increasing complexity of SOAs is handled in a completely distributed manner without the need of centralised or hierarchical components.

Organic Network Control

The *Organic Network Control* (ONC) system has been developed to dynamically adapt parameter configurations of data communication protocols to changing environmental conditions [24]. Typically, such parameters are buffer sizes, delay values, interval lengths, or counters. In traditional networking, these values are pre-configured using a static setup that works well on average. In contrast, the goal of the ONC system is to find the best parameter setting for each occurring situation. As one example, a broadcast protocol for mobile ad-hoc networks is adapted according to the movements of other nodes within the communication distance of the wireless communication infrastructure [25]. In principle, protocols from all layers of the protocol stack can be adapted, if they can be observed, modified, and evaluated using locally available information. Currently, research has focused on exemplary representatives from the domains of peer-to-peer networks, wireless sensor networks, and mobile ad-hoc networks. The ONC system is based on an adapted variant of the generic Observer/Controller architecture with a special focus on the 2-layered learning concept. The SuOC is one instance of a network protocol that can be observed locally and that is characterised by a set of variable parameters that can be altered at run-time. This altering process is performed by an Observer/Controller component situated at layer 1 that works on a set of rules with a fixed set of actions. Layer 2 is responsible for evolving new rules (the mapping between situations and protocol parameter configurations) in a simulation-based environment. Since each participant in the data communication network is equipped with an Observer/Controller component and no further hierarchical elements are contained in the resulting system, ONC can be considered as one example for a completely distributed system. Further details on organic network control are given in Cha. 6.1.11.

Organic Traffic Control

Signalised intersections in urban road networks are in the focus of the project *Organic Traffic Control* [17]. The project investigates an integrated traffic control system for urban environments and focuses in particular on the traffic-responsive control of traffic lights, on the self-organised coordination of signals, and on route guidance mechanisms.

The generic Observer/Controller architecture is applied at a signalised intersection where it evaluates and optimises the signalisation on-line. The Observer monitors the local traffic demands and evaluates the active signal plan's performance (e.g. with respect to the vehicular delay), while the Controller optimises the signalisation by learning new signal plans. Internally, the controller combines on-line reinforcement learning on Layer 1 and simulation-based optimisation on Layer 2, thereby being a successful example of 2-layered learning in a technical system.

Collaboration mechanisms that allow for the traffic-responsive coordination of intersections are a second focus of the project. The self-organised mechanisms establish progressive signal systems (also called *green waves*) in a road network. To assess the possibilities and limitations of decentralised traffic control systems, a completely decentralised mechanism (like in Fig. 3(b)) and a hierarchical variant (like in Fig. 3(c)) have been developed and compared [26].

The project's current focus is an integrated traffic management system that provides route guidance information to the drivers. Route recommendations are based on traffic demands measured at the network's intersections and on a local communication between the intersection controllers. Recent results of organic traffic control are summarised in Cha. 5.1.

4.3 Multi-levelled Observer/Controller Components

Multi-levelled OC systems consist of distributed Observer/Controller components that are hierarchically organised. Observer/Controller components on a higher level influence organic subsystems on lower levels (see Fig. 3(c)). The predominant application area are technical compounds with several subsystems that are sufficiently complex to require their own Observer/Controller. To achieve system-wide objectives, higher level Observer/Controller components influence groups of locally or logically related subsystems based on aggregated data gathered from lower levels. Examples include traffic networks with regional traffic light control (see Cha. 5.1) or smart-home environments with several controlled appliances.

MeRegioMobil

To adapt the consumers' energy demand to the power generation in the grid, the project *MeRegioMobil* investigates a smart-home environment equipped with several household appliances and an electric vehicle [2]. Within the smart-home environment, the vehicle's charging periods and the operation of various

appliances (like the washing machine or the freezer) are automatically rescheduled. The rescheduling is based on price signals that reflect a load-prediction of the energy grid, but considers user constraints (like a deadline for finishing a washing programme).

Rescheduling is performed by a multi-levelled Observer/ Controller framework: Each appliance is equipped with a local Observer/Controller component that observes the appliance's current state and can turn the appliance on or off according to current conditions. The local Observer/Controller components communicate their data (e.g. power consumption profiles) to a higher-level smart-home management device that centrally derives timing strategies for the smart-home. Chapter 6.1.9 presents more details about the smart-home environment and its multi-levelled observer/controller architecture.

5 Conclusion

Driven by the initial motivation and corresponding vision of OC, this article discussed a generalised design pattern for OC systems. Based on the insight that the emerging ubiquitous systems consisting of flexible networks of smart objects will no longer be manageable with current methods of systems engineering, a paradigm shift in system development is needed towards a generalised way to design OC-like systems. Therefore, the generic Observer/Controller architecture has been presented that incorporates several necessary mechanisms for OC systems like adaptivity, robustness, self-improvement in terms of learning, and flexibility in case of changing goals. Based upon the general design pattern, we introduced different distribution variants in order to cope with varying types of applications ranging from isolated to highly interconnected and complex systems.

To demonstrate the status of system design based on the principles of OC, this article presented current research projects that are built upon the proposed design pattern. Future work related to architectures of OC systems will proceed along two main aspects: the further refinement of the design pattern and the development of market-ready OC systems. The refinement of the design pattern and its contained components will consider characteristics like robustness and flexibility as well as an adaptation of the observation model at run-time due to some stimuli. Furthermore, the design pattern will find its way into market-ready systems. According to the initial vision for OC as postulated in 2003, we are moving towards OC-like applications ranging from smart production (like factories or warehouses) over networked systems (like data communication, smart grids, or urban traffic systems) to support systems for the daily life (like household robots or assisted living systems) [27]. Within the last six years of research on OC, a big step forward has been taken for some of these ideas (like urban traffic control systems), but others remain open for ongoing work.

References

1. L. R. Al-Sharif. Bunching in lifts: Why does bunching in lifts increase waiting time? *Elevator World*, 11:75–77, 1996.
2. B. Becker, F. Allerding, U. Reiner, M. Kahl, U. Richter, D. Pathmaperuma, H. Schmeck, and T. Leibfried. Decentralized energy-management to control smart-home architectures. In C. Müller-Schloer, W. Karl, and S. Yehia, editors, *Architecture of Computing Systems – ARCS 2010*, volume 5974 of *LNCS*, pages 150–161. Springer, 2010.
3. J. Branke, M. Mnif, C. Müller-Schloer, H. Prothmann, U. Richter, F. Rochner, and H. Schmeck. Organic Computing – Addressing complexity by controlled self-organization. In T. Margaria, A. Philippou, and B. Steffen, editors, *Proc. 2nd Int. Symp. on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006)*, pages 185–191. IEEE, 2006.
4. M. V. Butz. *Rule-Based Evolutionary Online Learning Systems – A Principled Approach to LCS Analysis and Design*. Springer, 2005.
5. E. Cakar, J. Hähner, and C. Müller-Schloer. Investigation of generic observer/controller architectures in a traffic scenario. In H.-G. Hegering, A. Lehmann, H. J. Ohlbach, and C. Scheideler, editors, *INFORMATIK 2008: Beherrschbare Systeme – dank Informatik*, volume 134 of *LNI*, pages 733–738. Köllen Verlag, 2008.
6. E. Cakar, M. Mnif, C. Müller-Schloer, U. Richter, and H. Schmeck. Towards a quantitative notion of self-organisation. In *Proc. IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 4222–4229. IEEE, 2007.
7. A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2nd edition, 2007.
8. J. J. Grefenstette and C. L. Ramsey. An approach to anytime learning. In *Proc. 9th Int. Workshop on Machine Learning*, pages 189–195, 1992.
9. J. O. Kephart and D. M. Chess. The vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.
10. L. Liu and H. Schmeck. Enabling self-organising service level management with automated negotiation. *IEEE/WIC/ACM Int. Conf. on Web Intelligence and Intelligent Agent Technology*, pages 42–45, 2010.
11. L. Liu, S. Thanheiser, and H. Schmeck. A reference architecture for self-organising service-oriented computing. In U. Brinkschulte, T. Ungerer, C. Hochberger, and R. G. Spallek, editors, *Architecture of Computing Systems – ARCS 2008*, volume 5974 of *LNCS*, pages 205–219. Springer, 2008.
12. C. Lode, U. Richter, and H. Schmeck. Adaption of XCS to multi-learner predator/prey scenarios. In *Proc. 12th Annual Conf. on Genetic and Evolutionary Computation (GECCO 2010)*, pages 1015–1022. ACM, 2010.
13. M. Mnif. *Quantitative Emergenz: Eine Quantifizierungsmethodik für die Entstehung von Ordnung in selbstorganisierenden technischen Systemen*. PhD thesis, Leibniz Universität Hannover, Institute for Systems Engineering, System and Computer Architecture Group, 2010.
14. M. Mnif and C. Müller-Schloer. Quantitative emergence. In *Proc. 2006 IEEE Mountain Workshop on Adaptive and Learning Systems (SMCals 2006)*, pages 78–84. IEEE, 2006.
15. C. Müller-Schloer. Organic Computing: On the feasibility of controlled emergence. In *Proc. 2nd Int. Conf. on Hardware/Software Codesign and System Synthesis*, pages 2–5, 2004.

16. D. Pathmaperuma. Lernende und selbstorganisierende Putzroboter. Master's thesis, Institut AIFB, Univ. Karlsruhe (TH), 2008.
17. H. Prothmann, J. Branke, H. Schmeck, S. Tomforde, F. Rochner, J. Hähner, and C. Müller-Schloer. Organic traffic light control for urban road networks. *Int. J. Autonomous and Adaptive Communications Systems*, 2(3):203–225, 2009.
18. O. Ribock, U. Richter, and H. Schmeck. Using organic computing to control bunching effects. In U. Brinkschulte, T. Ungerer, C. Hochberger, and R. G. Spallek, editors, *Architecture of Computing Systems – ARCS 2008*, volume 4934 of *LNCS*, pages 232–244. Springer, 2008.
19. U. Richter. *Controlled Self-Organisation Using Learning Classifier Systems*. PhD thesis, Universität Karlsruhe (TH), 2009.
20. U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, and H. Schmeck. Towards a generic observer/controller architecture for organic computing. In C. Hochberger and R. Liskowsky, editors, *Informatik 2006 – Informatik für Menschen*, pages 112–119. Köllen Verlag, 2006.
21. H. Schmeck. Organic Computing – A new vision for distributed embedded systems. In *Proc. 8th IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 201–203, 2005.
22. R. S. Sutton and A. G. Barto. *Reinforcement learning – An introduction*. MIT Press, 1998.
23. S. Thanheiser, L. Liu, and H. Schmeck. Towards Collaborative Coping with IT Complexity by Combining SOA and Organic Computing. *International Transactions on Systems Science and Applications*, 5(2):190–197, 2009.
24. S. Tomforde, E. Cakar, and J. Hähner. Dynamic Control of Network Protocols – A new vision for future self-organised networks. In J. Filipe, J. A. Cetto, and J.-L. Ferrier, editors, *Proc. 6th Int. Conf. on Informatics in Control, Automation, and Robotics (ICINCO'09)*, pages 285–290. INSTICC, 2009.
25. S. Tomforde, B. Hurling, and J. Hähner. Dynamic control of mobile ad-hoc networks – Network protocol parameter adaptation using organic network control. In J. Filipe, J. A. Cetto, and J.-L. Ferrier, editors, *Proc. 7th Int. Conf. on Informatics in Control, Automation, and Robotics (ICINCO'10)*, pages 28–35. INSTICC, 2010.
26. S. Tomforde, H. Prothmann, J. Branke, J. Hähner, C. Müller-Schloer, and H. Schmeck. Possibilities and limitations of decentralised traffic control systems. In *WCCI 2010 IEEE World Congress on Computational Intelligence*, pages 3298–3306. IEEE, 2010.
27. VDE/ITG/GI. Positionspapier Organic Computing, 2003.
28. M. Wünsche, S. Mostaghim, H. Schmeck, T. Kautzmann, and M. Geimer. Organic computing in off-highway machines. In *2nd Int. Workshop on Self-Organizing Architectures (SOAR 2010)*, pages 51–58. ACM, 2010.