Organic Computing – Addressing Complexity by Controlled Self-organization

Jürgen Branke[1], Moez Mnif[2], Christian Müller-Schloer[2], Holger Prothmann[1],
Urban Richter[1], Fabian Rochner[2], and Hartmut Schmeck[1]
[1]Universität Karlsruhe (TH) – Institute AIFB
76128 Karlsruhe, Germany
Email: {jbr|hpr|uri|hsch}@aifb.uni-karlsruhe.de
[2]Leibniz Universität Hannover – Institute of Systems Engineering
Appelstr. 4, 30167 Hannover, Germany
Email: {mnif|cms|rochner}@sra.uni-hannover.de

***Abstract*— In the past, the focus of the computer industry has been to improve hardware performance and add more and more features to the software. As a result, more and more appliances surrounding us are equipped with embedded computational power and wireless communication. As such, they become ever more flexible and multifunctional, and almost indispensable in daily life. On the other hand, the resulting systems become increasingly complex and unreliable, posing new challenges to designer and user.**

Organic Computing (OC) has the vision to address the challenges of complex distributed systems by making them more life-like (organic), i.e. endowing them with abilities such as self-organization, self-configuration, self-repair, or adaptation. The designer's task is simplified, because it is no longer necessary to exactly specify the low-level system behavior in all possible situations that might occur, but instead leaving the system with a certain degree of freedom which allows it to react in an intelligent way to new situations. Also, use of such systems is simplified, as they can be controlled by setting few high-level goals, rather than having to manipulate many low-level parameters with unclear influence.

In this paper, we give a general introduction to OC, and propose a generic observer-controller architecture as a framework for designing OC systems. Then, it is shown how to use this architecture at the example of a traffic light controller. The paper concludes with a summary and a discussion of future challenges.

I. Introduction

The impressive progress in computing technology over the past decades has not only led to an exponential increase in available computing power, but at the same time shrunk the chips to a miniature format. While only 20 years ago, the predominant computing platform was a company mainframe shared by many users, today, we are surrounded in everyday life by a multitude of embedded computing devices, including PDA, cell phone, digital camera, navigation system, MP3-player etc. An additional trend during recent years is that these devices are equipped with (often wireless) communication interfaces, allowing them to interact and exchange information.

But also other large technical systems consist of more and more interconnected electronic devices. For example, in cars, numerous processors and embedded systems keep the vehicle on road, control the engine with respect to combustion and pollution, assist the driver, provide security with air bags and seat belt systems, provide functions as air conditioning, navigation, parking assistant, information services and entertain the passengers. All these controllers are connected to a complex communication network. And this development has not stopped.

We are only a stone's throw away from scenarios like smart factory, with flexible robots self-organizing to satisfy the needs at hand, or smart cars that adapt to different drivers and road conditions, communicate with other cars on special events, or integrate personal devices (PDA, mobile telephone, notebook...) into their network.

While this development is exciting, the resulting systems become increasingly complex, up to the point where they can no longer be designed nor used easily. Already today, in the automotive sector, it is estimated that about half of all car break-downs are caused by electric and electronic components. E. g. in 2005 weak car batteries head the table of causes for break-downs listed by the ACE Auto Club Europa with 25% [1], while electronic components are listed on third position currently with 13% and still increasing their percentage.

Thus the question arises, how we will be able to design such complex distributed and highly interconnected systems, and how to make them reliable and usable. Clearly, the designer is not able to foresee all possible system configurations, and to prescribe proper behaviors for all cases. Also, we need to relieve the user from having to control in detail all parameters of the system, allowing him or her to influence the system on a higher level, e. g. by setting goals.

Organic Computing (OC) has the vision to address complexity by making technical systems more life-like, and endowing them with properties such as self-organization, self-configuration, self-repair, or adaptation [2].

In this paper, we will first give an overview on OC and a corresponding priority program of the German Research Foundation (DFG). In Section III we propose a generic observer-controller architecture with the goal to provide a generic template for implementing OC systems. Then, we show how to use this architecture at the example of a traffic light controller in Section IV. The paper is concluded with a summary and

978-0-7695-3071-0/07 $25.00 © 2007 IEEE
DOI 10.1109/ISoLA.2006.19

IEEE computer society

discussion of future challenges in Section V.

II. Organic Computing

As has been outlined above, the increasing complexity of technical systems calls for new design principles. It is impossible for a designer to foresee all possible configurations and to explicitly specify the entire behavior of a complex system on a detailed level. In particular if the system consists of many interacting components, it may exhibit new, emergent properties that are very difficult to anticipate. Emergent phenomena are often identified when the global behavior of a system appears more coherent and directed than the behavior of individual parts of the system (the whole is more than the sum of its parts). More generally, such phenomena arise in the study of complex systems, where many parts interact with each other and where the study of the behavior of individual parts reveals little about system-wide behavior. Especially in the area of multiagent systems emergence and self-organization are extensively studied, see [3], [4] for two recent surveys.

Despite their complexity, living creatures are very robust and have the natural ability to learn and adapt to an uncertain and dynamic environment. The idea of OC is therefore to address complexity by making technical systems more life-like and to develop an alternative to the explicit total a priori specification of a system by allowing the system to adapt and self-organize. OC systems should be designed with respect to human needs, have to be trustworthy and robust, adaptive, and flexible. They will exhibit so called self-x-properties as postulated for Autonomic Computing (AC) [5]–[7]: self-configuration, self-optimization, self-healing, self-explanation, and self-protection. Such systems would be able to learn about their environment over time, survive attacks and breakdowns, adapt to their users, and react sensibly even if they encounter a new situation for which they have not been explicitly programmed. AC pursues very similar goals, but so far focuses on server architectures as application, while OC focuses on distributed, self-organizing technical systems.

To achieve the goals of OC, technical systems need to have some degrees of freedom. While this has apparent benefits and will relieve the designer from specifying every single detail of the low-level system behavior, there are also some drawbacks: In order to learn, a system has to be allowed to make some errors, and it may react more slowly. Also, its behavior is less predictable, and allowing the system to self-organize possibly makes it more difficult to control.

Thus, one has to find a balance between creative self-organized bottom-up processes, and top down control. To this end, an observer-controller architecture has been proposed in [8]–[11], similar to the MAPE (monitor-analyze-plan-execute) cycle from AC (see also Figure 1 and Figure 2). The observer can monitor the overall system and detect problems early on. If an error is detected or anticipated, the controller will attempt to apply appropriate counteractive measures. This allows for self-organization but at the same time enables adequate reactions to control the – sometimes completely unexpected – emerging global behavior of these self-organized technical systems.

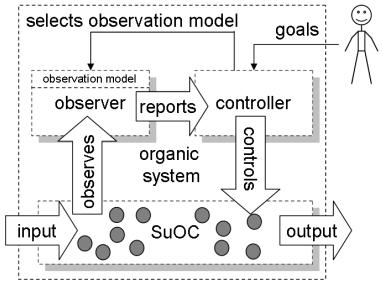


Fig. 1. Observer/controller architecture

The OC vision has first been formulated in the position paper of the Section of Computer Engineering ("Technische Informatik") of the "Gesellschaft für Informatik" (GI) [12]. In 2005, the German Research Foundation (DFG) approved a priority research program on OC. This research program addresses fundamental challenges in the design of OC systems; its objective is a deeper understanding of emergent global behavior in self-organizing systems and the design of specific concepts and tools to construct and control OC systems for technical applications. Topics such as adaptivity, reconfigurability, emergence of new properties, and self-organization play a major role. Currently the research program provides funding for 18 research projects with a total volume of around 2 million EUR per year. The topics of these projects range from traffic control over robot coordination to chip design. More information on the different projects can be found at [13]. The remainder of this paper discusses results from two of these projects, namely "Quantitative Emergence", with the goal to develop a generic observer-controller architecture, and "Organic Traffic Control", which applies ideas from OC to the design of traffic light controllers.

Overall, OC is not yet an established research area, but an exciting, interdisciplinary field offering a vision on how to manage the increasing complexity of technical systems by allowing for controlled self-organization.

III. A Generic Architecture for Organic Computing Systems

A. The Observer/Controller Paradigm

In order to assess the behavior of the technical system and – if necessary – for a regulatory feedback to control its dynamics, we assume that a generic observer/controller architecture is required as depicted in Figure 1 [8]. The observer/controller has a set of sensors and actuators to measure system variables and influence the system. Together with the system under observation and control (SuOC), the observer/controller forms what we call an "organic system".

Generally, we assume that the SuOC consists of a large collection of communicating active objects possessing certain common attributes. The observer collects data from the system (micro and macro level properties) and computes some indicators characterizing the global state and the dynamics of the system. We call these indicators "situation parameters".

The controller compares the situation parameters with the goal defined by the user, and decides whether an intervention is required and, if so, what action would be most appropriate. The controller may affect the SuOC by influencing:

- the local decision rules of the SuOC elements,
- the system structure including e. g. the communication between the SuOC elements or the number of elements, or
- the environment, which will indirectly influence the system by changing the data observed by the SuOC elements through their local sensors.

It is important to note that an organic system continues to work and does not break down if observer and controller stop working. The controller interferes only when necessary and affects only some system parameters, it does not control single elements in detail.

B. Generic Architecture

Figure 2 shows the architecture in more detail, and also illustrates the workflow. The observation process consists of the steps monitoring, pre-processing, data analysis, prediction, and aggregation. An observation model customizes the observer by selecting the observable attributes of the system, the analysis detectors, and the appropriate prediction methods.

The monitor samples the attributes of the SuOC according to a given sampling rate. The observed system data consists of individual data on the level of single elements, and some global system attributes. All measured data is stored in a log file for every loop of observing/controlling the SuOC. This stored data can be used within the predictor or for the calculation of time-space-patterns in the data analyzer.

In the pre-processing step some derived attributes can be computed from the raw data (e. g. an attribute velocity can be derived from attributes x-coordinate and y-coordinate collected in subsequent time steps). The data analyzer applies a set of detectors to the pre-processed data vector. This could be a computation of clustering, or emergence parameters with respect to the definition in [14], or some other mathematical and statistical methods. At the end of this step a system-wide description of the current state is provided. In a more or less parallel step the predictor processes the data coming from the pre-processor and the data analyzer and tries to predict a future system state. It can use its own methods or some methods of the data analyzer combined with prediction methods taken e. g. from technical analysis. The results of data analyzer, predictor, and possibly some unprocessed data coming from the pre-processor are handed on to the aggregator. These so called situation parameters are transmitted to the controller.

As described in Section III-A, the controller's task is to guide and control the self-organization process between the elements based on the data received from the observer. It will interfere only when necessary. At the heart of the controller is a decision module called action selector, which selects the action A_i that is most appropriate for the current situation. It has to make this decision quickly to react in real-time. On top of this component, the adaptation module implements

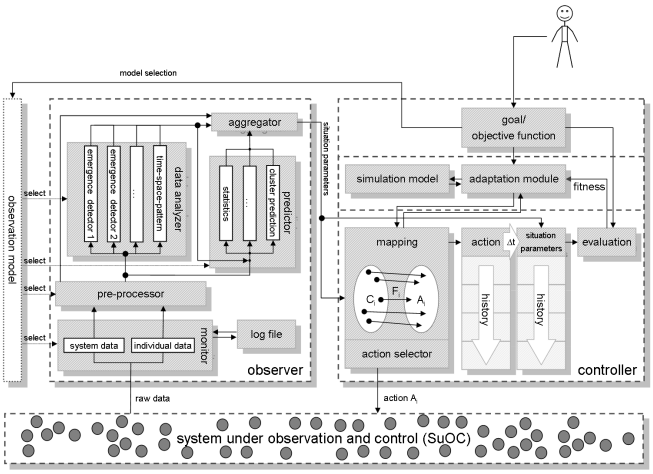


Fig. 2. Generic observer/controller architecture

some learning and planning capabilities. The learning uses an evaluation of implemented actions to improve the mapping from situation to actions. Evaluation is performed based on a database with history data, allowing to attribute system changes to specific actions. Similar to an approach called "Anytime Learning" proposed be Greffenstette and Ramsey [15], the adaptation module may be connected to an additional simulation model, which allows it to approximate the consequences of specific actions in specific simulated environments before testing them on the real system. This module can also be used to design or plan completely new actions, e. g. by optimizing action parameters for a specific situation. The adaptation aims at a system behavior that more closely matches the system goals formulated by the developer or current user.

C. Realization

The generic architecture has to be customized to different scenarios by adapting the various components of the observer (including the observation model) and the controller. The architecture depicted in Section III-B can be realized in the following ways (see Figure 3):

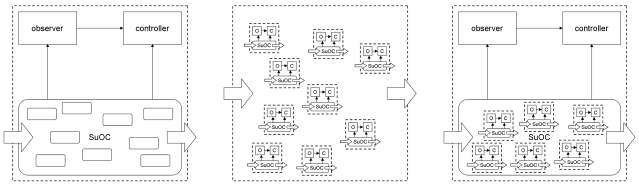
(a) central: An observer/controller for the whole technical system.
(b) decentral: An observer/controller on each system element.
(c) multi-level: An observer/controller on each system element as well as one for the whole technical system. In this way holarchies[1] could be built.

The choice of the appropriate observer/controller realization is a design decision that has to be done by the developer in the design phase of the technical system.

IV. Organic Traffic Control

In this section, we describe how to implement the generic OC architecture on a challenging and highly relevant technical

[1]A holarchy is a hierarchy of holons, where a holon is a whole and at the same time a part of a whole [16].



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

Fig. 3. Observer/controller realization

problem: the control of an urban traffic network.

The control of a junction so as to minimize waiting times has been shown to be an NP-complete problem [17]. In practice, a large number of different control systems are used to manage traffic networks, the two main approaches being the following: On the one hand, neighboring junctions running fixed-time programs can be closely coordinated to form synchronized traffic lights ("green waves"). This pre-programmed approach has the drawback of not being responsive to traffic. On the other hand, traffic-responsive controllers allow an adaptation to changes in the traffic flow, but the coordination of neighboring junctions is then very hard.

Recently, research focuses on a third approach – the coordination of traffic-responsive controllers through principles of self-organization. Promising results in this area were presented e. g. in [18] where a fluid-dynamic model for the simulation of urban traffic networks was used to investigate self-organization principles for traffic control.

The "organic approach" to traffic control discussed in this section belongs to the category of self-organizing traffic systems. We propose to use traffic-responsive junction controllers with overlapping sensor horizons (see Figure 4) to control the network. By using data provided by neighboring junctions for local optimization, a sort of implicit global optimization is expected. To keep the complexity of the architecture at a manageable level currently only a single junction is considered. As work progresses the controlled network will get larger and more complex, thus data exchange between controllers will be incorporated. It still has to be investigated if the controller network can be realized in a completely decentralized way. A multi-level realization with coordinating components for different subnetworks might be helpful to achieve global optimization.

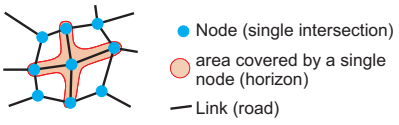


Fig. 4. Sensor horizon of junction controllers

In both cases, fast reaction to changing traffic conditions is a vital prerequisite to reach good performance. But since the junction controllers should be kept as generic as possible, they have to adapt to the environment they are placed in, i. e. they have to learn. To learn means to make mistakes, which is detrimental to performance. To cope with this challenge the necessary functionality of a junction controller is realized as an organic system.

The SuOC in this traffic scenario consists of a signaled junction equipped with detectors, traffic lights, and a parameterizable traffic light controller (TLC). The TLC can act instantly on small changes in the traffic situation by slightly varying its phase timings within predefined boundaries and in a predefined way. Its behavior is defined by a set of parameters that have to be adapted whenever the traffic situation changes significantly.

Parameter adaptation is performed by an implementation of the generic observer/controller architecture presented in Section III. While the observer is responsible for analyzing the traffic situation, the controller decides when and how to change the parameter set for the TLC. Therefore, the controller maps traffic situations to parameter sets and keeps track of how well each parameter set performs. Alternative parameter sets are generated using an Evolutionary Algorithm (EA) and a simulation of the traffic network, adjusted to the traffic situation in question, as testbed. This way, trial-and-error search in the real traffic network can be avoided. The internals of the adapted generic observer-controller-model for the traffic scenario are depicted in Figure 5 and are discussed in the following sections.

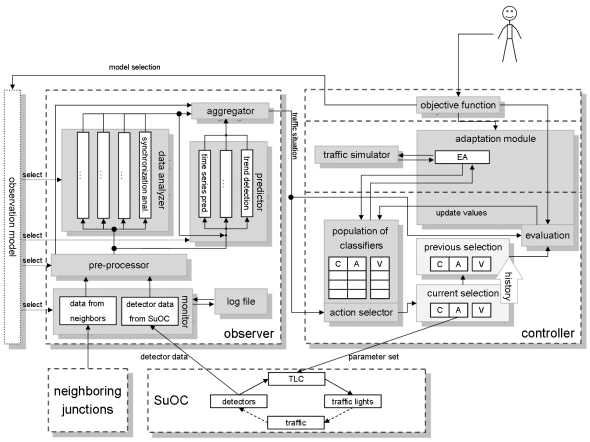


Fig. 5. Observer/controller architecture adapted to traffic control

A. Observer Functionality

The task of the observer component in the traffic scenario is to analyze the traffic situation and generate macro-level system data. The detector measurements obtained in the SuOC constitute the raw data available to the observer to perform this task (i. e. individual data). Different types of detectors – like inductive loops, infrared, or video detectors – might be installed in the SuOC, all exhibiting different measurement capabilities. These include, among others, detecting the presence of vehicles, the recognition of vehicle types, counting capabilities as well as speed, headway and density measurements. All this data is collected by the monitor and stored in

the observer's log file. When a network of several junctions is considered, additional information from neighboring junctions will be available to the observer (corresponding to system data).

In a pre-processing step, the collected data will be used to calculate the traffic flows (measured in vehicles per time unit) for all signal groups installed at the junction. When studying larger traffic networks, additional data analyzing will be necessary to detect effects of the synchronization of different junctions. Furthermore, a prediction component can be used to discover trends in the traffic development or to make predictions by comparing historical time series with the currently observed measurements.

Currently, no further data analysis or prediction techniques are used following the pre-processing. The traffic flows calculated in the pre-processing component are simply aggregated into a vector representing the current traffic situation. The detected traffic situation is the analogy to the situation parameters in the generic architecture. It can be visualized as depicted in Figure 6.

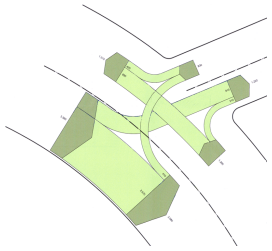


Fig. 6. Visualization of a traffic situation detected by the observer. Arrow width is proportional to traffic flow.

B. Controller Functionality

As said before the controller uses some kind of mapping to choose an appropriate parameter set to configure the TLC for each traffic situation presented as input by the observer. To keep the problem manageable, the input space in terms of different traffic situations has to be partitioned so that situations which are sufficiently similar to allow usage of the same parameter set will be covered by the same mapping entry. Since it is hard to anticipate all traffic situations that might occur at the controlled junction and since it would need a tremendous effort of a traffic engineer to develop good control strategies for all situations that can be imagined, the means for generating mapping entries have to be provided by the controller. These tasks combined are what Learning Classifier Systems (LCS) [19] are supposed to do: classify input, find appropriate action, learn by gaining experience.

A standard LCS uses rules (called classifiers) to store its knowledge. These classifiers consist of a condition that is matched against the input, an action and some kind of fitness or value. The value is used to decide which classifier to choose if more than one matches the current input. The encoding of conditions is done such that different levels of "generality" are possible, hence the range of input values each classifier matches against may vary from a single point to the entire search space. New classifiers are generated using genetic operators like crossover and mutation on existing ones, changing both condition and action (this is called "rule discovery"). Furthermore, every time no classifier matching the current input is available, one or more classifiers with a matching condition and randomly chosen action are created (this is called "covering").

After a classifier has been generated, the system has to determine its value. Every time a classifier's action is chosen, the value of this classifier is updated using some objective function. Usually, the environment's reaction on some action executed by the LCS is not instantaneous, therefore the current value of the objective function is used to update the value of the classifier that was active during the *previous* time step.

The simplest way to increase performance would be to try and find classifiers which maximize the objective function's value. Another approach more suitable for complex problems is to use three distinct components as "value": the prediction of the objective function's value after this classifier's action has been executed, the average error of this prediction, and a fitness value computed from this prediction error. The prediction is used to choose an action for execution: The classifier with best prediction, weighted by its fitness, is likely to result in best performance. Furthermore, the fitness is used to find classifiers suitable as input for the generation of new classifiers using genetic operators. The goal is to represent the entire search space with as few classifiers as possible while keeping only those that are more *accurate* in their prediction. This approach is realized in XCS, the eXtended Classifier System [20].

Classifier systems like XCS have been applied to problems related to traffic control in the past. In [21] an LCS was used to control a simplified junction by continuously changing green times based on observed queue lengths. During an exploration period, the system evolved a classifier population that afterwards was applied to control the junction without further learning. However, the traffic network used in this paper is noticeably more complex.

Our goal is the development of an architecture usable for traffic networks of realistic complexity. Obviously, the mechanism used for the generation of knowledge is of crucial importance. As it is not a viable approach to let a system learn using a real traffic network as testbed, a modified classifier system similar to XCS is used, but with its functionality divided between two components: a basic classifier system and a separate component for rule discovery. This separate component utilizes a model of the real traffic network running on a simulator to search for optimal TLC parameter sets. Such a parameter set together with the encoded situation it was optimized for and its value forms a classifier that is then fed into the population of the classifier system. Classifiers from this population are applied by the classifier system in the real network and their valuation is further improved based on feedback from reality. Both components (basic classifier system and rule discovery) use the same objective function

to update the value of a classifier or to find the optimal parameter set, respectively. Evaluation criteria incorporated into the objective function might be the average waiting time or the average number of stops.

Since the explorative behavior of the system is moved to an external component, the task of the basic system is to find the best action among those available. This is simple if classifiers exist that match the current input: The classifier is chosen, whose prediction is best. If no classifier matches, a new matching one has to be generated. The proposed approach is to choose the classifier whose condition is closest to the current input, copy it, widen its condition just enough to match and discount its value slightly. This way a quick response is possible (waiting for a new solution to be generated using the simulator would take too long) and the probability that the chosen action will perform acceptably is greater than if the action was chosen randomly. If this widening exceeds a certain threshold, the current situation is passed simultaneously to the rule discovery component so that, when a similar situation is encountered again, a specifically optimized classifier is available.

Within the rule discovery component, an EA is responsible for producing new classifiers. It generates populations of TLC parameter sets and evaluates their applicability for the current situation using a simulation model. The parameter sets are used to configure a simulated TLC that controls the simulated junction for a defined period of time. After a warm-up period allowing the traffic to build up, a number of quality measures (like the average waiting time or the average number of stops) are recorded and used by the EA by means of the objective function to evaluate the different parameter sets in the same way as action values are updated after execution in real traffic. At the end of this process, a new classifier is created that maps the simulated situation to the best parameter set found by the EA. The classifier's value is initialized based on the simulation results.

By using a simulation model to generate new classifiers, bad TLC parameter sets can be identified without negative consequences for the SuOC. Furthermore, the employment of a simulation software allows the fast evaluation of many parameter sets, which should result in an improved learning speed of the architecture. A potential drawback of this simulation-based approach might be differences between simulation and reality. But since microscopic traffic simulators available today are quite accurate after their calibration [22], we expect no major problems here. Residual errors in the valuation of a new classifier are corrected when the classifier is applied in the SuOC and its true quality can be observed.

Now the architecture described above matches quite nicely the generic observer/controller architecture shown in Section III. The classifier population performs a mapping of traffic situations to adequate TLC parameter sets. The action selector matches the situation detected by the observer against the available classifiers and selects – if necessary – among several applicable actions (i. e. among several TLC parameter sets) based on their values. Whenever no matching classifier is available, the classifier whose condition matches the observed situation most closely is selected and the rule discovery (adaptation) component is used in combination with the simulation model to create a new classifier based on simulation results. The selected parameter set is then applied as action in the SuOC. With the next controller invocation, the previously selected action is evaluated based on the objective function of the controller and an appropriate value update is performed.

V. Conclusion and Challenges

This paper has given a brief introduction to the German priority research program on Organic Computing. One of the major arguments driving the vision of Organic Computing is 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. In particular, at design time, it will not be possible to explicitly design appropriate reactions to all the possible changes in the environment and, furthermore, there is the possibility of emerging global behavior due to unanticipated interactions between active components. Therefore, as outlined in this paper, these systems need the capability to organize several aspects of their behavior in an autonomous way, independent of explicit external interference. They have to exhibit various self-x properties, which should enable them to adapt to changes in their environment in order to show robust behavior (i. e. maintain a requested functionality in spite of changing environmental parameters) or to behave in a flexible way by modifying their behavior in response to changing requirements. The observer/controller architecture has been suggested as a generic approach to solve these problems by providing the feedback, decision and learning capabilities which are necessary for realizing self-x properties. Furthermore, this architecture allows for adequate external control of self-organized behavior using appropriate higher level objective functions within multi-level OC-architectures. The control of urban traffic lights served as an example for showing how the generic architecture may be instantiated for a specific application.

Hence, the claim of Organic Computing is to provide an appropriate response to the design and management challenges of our anticipated, digitally enhanced environment. But, obviously, Organic Computing will only be a valid approach to solving the management problems of these complex systems if we can argue formally and give empirical evidence that the behavior of an organic system will be more reliable, more robust, more flexible, and more trustworthy than a system without "organic" properties. Therefore, there is a need for empirical studies of organic computing systems in various application domains. Furthermore, the properties of these systems should be investigated by formal methods. In particular, it would be important to find ways to proof and validate the proper functionality of learning, self-organizing systems. Other challenges refer to the scalability of system properties and to the generation of trust in the behavior of organic computing systems: The potential of self-organizing,

adaptive systems in our personal environment will only be exploited, if these systems show acceptable behavior, and an important prerequisite for acceptance is the development of trust.

ACKNOWLEDGMENT

We gratefully acknowledge the financial support by the German Research Foundation (DFG) within the priority program 1183 “Organic Computing”.

REFERENCES

[1] ACE Auto Club Europa, “Schwache Batterie Pannenursache Nummer 1,” April 2006, visited October 2006. [Online]. Available: http://www.ace-online.de/md/12910-1420/ACE_Grafik_Autopannen_sw.pdf

[2] H. Schmeck, “Organic Computing - A new vision for distributed embedded systems,” in *Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*. IEEE Computer Society, May 2005, pp. 201–203.

[3] G. Di Marzo Serugendo, N. Foukia, S. Hassas, A. Karageorgos, S. Kouadri Mostéfaoui, O. F. Rana, M. Ulieru, P. Valckenaers, and C. Van Aart, “Self-organisation: Paradigms and applications,” in *Engineering Self-Organising Systems*, ser. LNAI, G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, Eds., vol. 2977. Springer, 2004, pp. 1–19.

[4] G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageorgos, “Self-organisation and emergence in MAS: An overview,” *Informatica*, vol. 30, no. 1, pp. 45–54, 2006.

[5] IBM, “Autonomic computing: Creating self managing computing systems,” 2003, visited October 2006. [Online]. Available: http://www-03.ibm.com/autonomic/

[6] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[7] R. Sterritt, “Autonomic computing,” *Innovations in systems and software engineering*, vol. 1, no. 1, pp. 79–88, March 2005.

[8] C. Müller-Schloer, “Organic Computing: On the feasibility of controlled emergence,” in *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2004)*, A. Orailoglu, P. H. Chou, P. Eles, and A. Jantsch, Eds. ACM, 2004, pp. 2–5.

[9] C. Müller-Schloer, C. von der Malsburg, and R. P. Würtz, “Organic Computing,” *Informatik Spektrum*, vol. 27, no. 4, pp. 332–336, 2004.

[10] T. Schöler and C. Müller-Schloer, “An observer/controller architecture for adaptive reconfigurable stacks,” in *Systems aspects in organic and pervasive computing – ARCS 2005*, ser. LNCS, M. Beigl and P. Lukowicz, Eds., vol. 3432. Springer, March 2005, pp. 139–153.

[11] U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, and H. Schmeck, “Towards a generic observer/controller architecture for organic computing,” in *INFORMATIK 2006 – Informatik für Menschen!*, ser. GI-Edition – Lecture Notes in Informatics (LNI), C. Hochberger and R. Liskowsky, Eds., vol. P-93. Bonner Köllen Verlag, October 2006, pp. 112–119.

[12] VDE/ITG/GI, “Positionspapier Organic Computing,” 2003, visited October 2006. [Online]. Available: http://www.gi-ev.de/fileadmin/redaktion/Presse/VDE-ITG-GI-Positionspapier_20Organic_20Computing.pdf

[13] DFG Priority Program 1183 Organic Computing, “SPP 1183 website,” 2005, visited October 2006. [Online]. Available: http://www.organic-computing.de/SPP

[14] M. Mnif and C. Müller-Schloer, “Quantitative emergence,” in *Proceedings of the 2006 IEEE Mountain Workshop on Adaptive and Learning Systems (IEEE SMCals 2006)*, July 2006, pp. 78–84.

[15] J. J. Grefenstette and C. L. Ramsey, “An approach to anytime learning,” in *Proceedings of the 9th International Workshop on Machine Learning (ML 1992)*, D. H. Sleeman and P. Edwards, Eds. Morgan Kaufmann, 1992, pp. 189–195.

[16] Wikimedia Foundation, “Holarchy – Wikipedia,” 2006, visited July 2006. [Online]. Available: http://en.wikipedia.org/wiki/Holarchy

[17] J. Mertz, “Ein mikroskopisches Verfahren zur verkehrsadaptiven Knotenpunktsteuerung mit Vorrang des öffentlichen Verkehrs,” Dissertation, Fachgebiet Verkehrstechnik und Verkehrsplanung der Technischen Universität München, München, 2001.

[18] D. Helbing, S. Lämmer, and J.-P. Lebacque, “Self-organized control of irregular or perturbed network traffic,” in *Optimal Control and Dynamic Games*, C. Deissenberg and R. F. Hartl, Eds. Springer, 2005, pp. 239–274.

[19] J. H. Holland, L. B. Booker, M. Colombetti, M. Dorigo, D. E. Goldberg, S. Forrest, R. L. Riolo, R. E. Smith, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, “What is a Learning Classifier System?” in *Learning Classifier Systems. From Foundations to Applications*, ser. LNAI, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds., vol. 1813. Springer, 2000, pp. 3–32.

[20] S. W. Wilson, “Classifier fitness based on accuracy,” *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[21] L. Bull, J. Sha’Aban, A. Tomlinson, J. D. Addison, and B. Heydecker, “Towards distributed adaptive control for road traffic junction signals using learning classifier systems,” in *Applications of Learning Classifier Systems*, L. Bull, Ed. Springer, 2004, pp. 276–299.

[22] H. Xiao, R. Ambadipudi, J. Hourdakis, and P. Michalopoulos, “Methodology for selecting microscopic simulators: Comparative evaluation of AIMSUN and VISSIM,” Department of Civil Engineering, University of Minnesota, Tech. Rep. CTS 05-05, 2005.