Towards a generic observer/controller architecture for Organic Computing

Urban Richter[1], Moez Mnif[2],
Jürgen Branke[1], Christian Müller-Schloer[2], Hartmut Schmeck[1]

[1]Institute of Applied Informatics
and Formal Description Methods
Universität Karlsruhe (TH)
D-76128 Karlsruhe, Germany
{uri,jbr,hsch}@aifb.uka.de

[2]Institute of Systems Engineering
Universität Hannover
Appelstr. 4
D-30167 Hannover, Germany
{mnif,cms}@sra.uni-hannover.de

Abstract: Technical scenarios in areas like automotive or production systems will increasingly consist of a large number of components cooperating in potentially unlimited and dynamically changing networks to satisfy the functional requirements of their execution environment. Due to the high complexity it will be impossible to explicitly design the behaviour of the components for every potentially arising situation. Therefore, it will be necessary to leave an adequate degree of freedom allowing for a self-organised behaviour. Organic Computing (OC) has developed the vision of self-organising systems adapting robustly to dynamically changing environments without running out of control. This paper focuses on the design of a generic system architecture which allows for self-organisation but at the same time enables adequate reactions to control the – sometimes completely unexpected – emerging global behaviour of these self-organised technical systems.

1 Introduction

In 1850 more than 700 French soldiers marched lock-step over the rope bridge of Angers. The bridge began to vibrate, collapsed, and 226 soldiers died. Today, it is not allowed to march lock-step over a bridge. This tragedy is often quoted as an example of a resonance catastrophe which denotes a situation where a building (or any other technical system) is destroyed by vibrations. It is based on eigenfrequency; energy is accumulated in the system due to periodic stimulation. The stored energy and the continuous energy addition excite the system und lead to its destruction by exceeding a system specific energy threshold. Vibration absorbers are used to protect a system and dissipate the energy.
Similar phenomena increasingly occur also in informatics. Analogous to the example above technical scenarios in the domain of informatics are becoming more and more complex and unmanageable. Our daily life provides various situations where we are surrounded by self-organising, interacting systems showing unknown and emergent behaviour.
Emergent phenomena are often identified when the global behaviour of a system appears

more coherent and directed than the behaviour 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 behaviour of individual parts reveals little about system-wide behaviour.
Based on these trends, the question is not whether complexity increases or informatics is confronted with emergence, but how we will design new systems that have the possibility to cope with the emerging global behaviour of self-organising systems by adequate control actions, e. g. by using analogous to vibration absorbers as seen above.
Organic Computing (OC) [DFG05] has developed the vision of self-organising systems adapting robustly to dynamically changing environments without running out of control. This paper focuses on the design of a generic system architecture (an observer/controller architecture) which allows for self-organisation but at the same time enables adequate reactions to control the – sometimes completely unexpected – emerging global behaviour of these self-organised technical systems. Our proposed architecture can be used in a centralised, distributed or multi-levelled way.
The paper is structured as follows: Section 2 outlines our motivation and provides highlights of the organic methodology. We describe our computational model, including input process, metrics for observation and control algorithms in Section 3 and 4. We outline possible future work and present concluding remarks in Section 5.

2 Observer/controller architecture

Looking at the anticipated information and communication technologies of the year 2010 and beyond [VDE03], we have to acknowledge the fact that we shall be surrounded by large systems of intelligent devices interacting and cooperating in potentially unlimited networks. The design complexity of these systems calls for new design paradigms, and due to the effects of interaction in dynamically changing environments the global behaviour of these systems might be unexpected [Sch05]. In particular, to some degree, these systems will organise themselves, independently of initial designs or external interventions. The major idea is to tame design complexity of technical systems by leaving considerable degrees of freedom for their structure and behaviour and by bestowing upon them some life-like characteristics, allowing them to learn and adapt with respect to dynamically changing environments. Such systems would be able to maintain themselves and would not need scores of humans to set them up and keep them running. They 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 programmed explicitly. OC defines an organic computer as a self-organised system that can adapt to a dynamically changing context and achieves the so called self-x-properties as postulated for Autonomic Computing [KC03, Ste05]: self-configuration, self-optimisation, self-healing, self-explanation, and self-protection. But in spite of being self-organised, an essential feature of organic systems will be their ability to react sensibly to external – in particular human – requirements and to allow for control actions that might be necessary to keep their behaviour within preferred regions of the configuration space

taking into account the effects of emergence.
As indicated above, emergent global behaviour is a key aspect of OC systems. We assume that one way to achieve the desired goals is to move from a centralised system to a decentralised system, consisting of a large number of interacting sub-systems. In order to assess the behaviour of such a 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 [MS04, MSvdMW04, SMS05]:

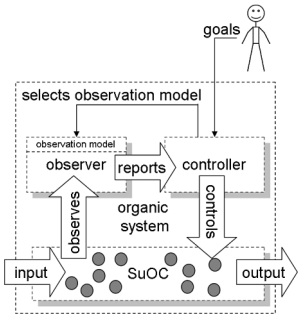

Figure 1: Observer/controller architecture

The decentralised system is termed system under observation/control (SuOC), the observer and the controller are responsible for an appropriate surveillance and feedback.
Sensors and actuators are at the heart of our organic architecture. On top of the SuOC a control loop is created. It observes behaviour through sensors, compares results with expectations, decides what action is necessary and controls the SuOC with the best known action through actuators.
To compare with expectations knowledge of historical and current data, rules and beliefs is assumed, i.e. this is not necessarily a trivial task. It is the observer's task to measure, quantify, and predict emergent behaviour with basic metrics. The observer collects and aggregates information about the SuOC. The aggregated values (system indicators) are reported to the controller who takes appropriate actions to influence the SuOC. The observation behaviour itself is variable. The observer model influences the observation procedure, e. g. by selecting certain detectors or certain attributes of interest. The feedback from the controller to the observer directs attention to certain observables of interest in the current context. Based on the aggregate results from the observer, the controller can benchmark the data with an objective function and either knows or learns which actions are best to guide the SuOC in the desired direction.
It is important to note that an organic system continues to work and does not break down if observer and controller stop working. Thus, the main objective of our architecture is to achieve a controlled self-organised behaviour. In comparison with classical system design, OC systems have the ability to adapt and to cope with some emergent behaviour for which they have not been programmed explicitly. In this paper we describe a centralised observer-controller architecture. The goal of OC is to build systems that perform their tasks by using (controlled) self-organisation. However, this is independent of using centralised or decentralised observer/controller architectures, since the elements of the system work autonomously and the controller affects some local control parameters only and does not control single elements in detail.

3 Observer

The aim of the observer is to perform an aggregation of available information about the SuOC in form of indicators to give a global description (called situation parameters) of

the state and the dynamics of the underlying system. The main tasks of the observer are basically: (a) identifying and characterising the nature of the phenomenon representing the current system status, and (b) predicting the future status of the system. Figure 2 outlines a generic observer architecture. The observer is guided by an observation model, which is responsible for the following tasks: (1) selection of observable attributes, (2) selection of appropriate analysis tools with regard to the purpose given by the controller and (3) selection of appropriate prediction methods.

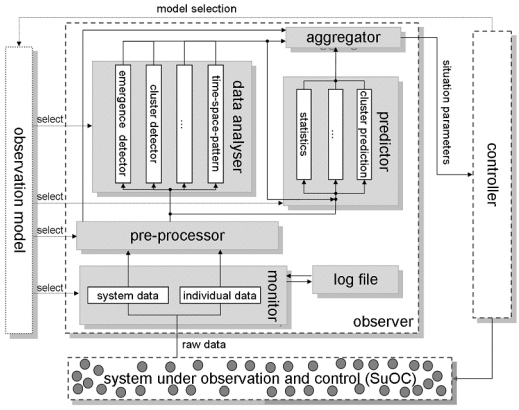

Figure 2: Generic observer architecture consisting of a monitor, a pre-processor, a data analyser, a predictor, and an aggregator.

The observation process involves the following steps and components:

- Monitor: The SuOC is considered as a set of elements possessing certain attributes (in terms of multi-agent systems). The monitor samples the attributes of the SuOC according to a sampling frequency given by the observation model. The information coming from the SuOC constitutes raw data (unprocessed) for the observer, which can be classified into individual data common to all elements of the system and some global system attributes. From a chronological point of view, monitoring the SuOC is nothing else than the generation of a time series, reflecting the current state of the system as well as its history. The sensory equipment of the SuOC limits the selection of observable attributes and the resolution of the measurement.

- Log file: 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 analyser.

- Pre-processor: In the next step (pre-processing) some derived attributes can be computed from the raw data. E. g. an attribute velocity can be derived from the attributes x-coordinate and y-coordinate taking into account the history of these two attributes. The pre-processing of the raw data includes also a selection of the relevant data which is required to compute aggregated system-wide parameters. The pre-processed data is passed to the data analyser and the predictor components.

- Data analyser: The data analyser applies a set of detectors to the pre-processed data vector. These detectors could be a kind of computation of clustering, emergence following the definition in [MMS06] or some mathematical and statistical values. At the end of this step a system-wide description of the current state is provided.

- Predictor: The predictor processes the data coming from the pre-processor and results coming from the data analyser with the objective of giving a prediction of the future system state. The predictor can use its own methods [MMS06] or some methods of the data analyser combined with prediction methods taken e. g. from technical analysis. Prediction involves an analysis of the system history. For this purpose the predictor is equipped with a memory to store a given time window. We are especially interested in the prediction of future behaviour in order to reduce the reaction time of the controller and – hence – to increase the probability to prevent unwanted behaviour in due time, or to perceive the success of a controller intervention at an early stage.

- Aggregator: The results of data analyser, predictor and possibly some raw data coming from the pre-processor are handed on to the aggregator. The aggregator also has a memory, in which current values as well as their history are stored forming a set of data vectors (one for each given result). These vectors are needed to perform filtering as e. g. a smoothing of the results to eliminate the effects of noise. The aggregator delivers a set of filtered current and previous values to the controller. This constitutes an abstract description of the current state and the dynamics of the SuOC.

The observer can be customised to different scenarios by adapting the observation model (selection of observable attributes and tools). Both, data analyser and predictor, can be regarded as a toolbox of observation methods.

4 Controller

We explore possibilities to influence the emergent behaviour of complex systems, assuming that the system consists of a large number of relatively simple, interacting elements/agents. There exist three possible objectives: (1) to influence the system such that a desired emergent behaviour appears, (2) to disrupt an undesired emergent behaviour as quickly and efficiently as possible and (3) to construct the system in a way such that no undesired emergent behaviour can develop.
It is the controllers' task to guide the self-organisation process between the elements, but to interfere only when necessary (e. g. when a system parameter exceeds a certain threshold). At least three general types of control can be identified to generate or disrupt emergent behaviour:

1. Influencing the local decision rules of the simple agents modifies the local behaviour of the individual.

2. Influencing the system structure: We assume that the elements base their actions on local information, where "local" is defined by a neighbourhood and an interconnection network. Modifying this network, in particular with respect to global characteristics, will change the global behaviour of the system. Also, changing the absolute number of elements influences neighbourhoods and at last the behaviour of the SuOC.

3. Influencing the environment allows indirect control of the SuOC and works only if system elements have sensors to measure and react to a modified environment, and the controller has actuators to influence the environment. In our theoretical discussion we should mention that the environment cannot be controlled at all and it is not clear where the environment of the SuOC starts and ends really. One possibility would be to declare some of the actuators to be part of the controller, not the SuOC. Then the controller can actually modify the environment to change the overall behaviour of the SuOC.

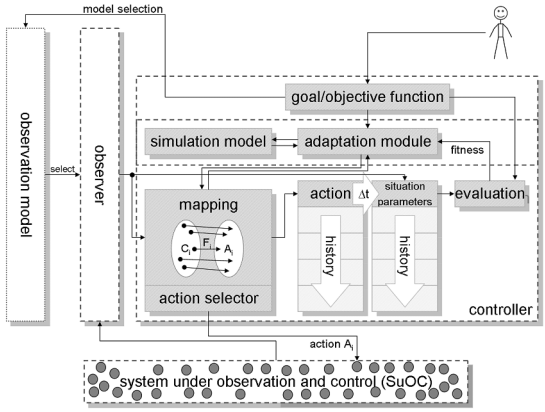

Figure 3: Generic controller architecture

In the following, we present a generic architecture of the controller for OC, which has to be customised to each individual scenario. The controller has three interfaces, see Figure 3: (1) The aggregated data are obtained from the observer. (2) The objectives are imposed on the controller by the user or developer using the second interface. This global objective function defines the goal of the controlled self-organising process and is also used for the evaluation routine of further actions. (3) The third interface contains all information needed for interaction and reconfiguration of the SuOC. Every controlled system provides a number of different parameters and interfaces for manipulation. This information is predetermined, depends on the scenario, and is not learned by the controller.
The decision module called action selector and the mechanisms of machine learning are the most important components. We distinguish three main loops of planning and learning: The first loop recieves the observed data, selects the best weighted action (A_i) that is most appropriate for the current situation (C_i), and the action selector forwards the chosen

action to the SuOC. This loop simply applies the best out of a given set of actions and does not involve learning. It is aimed at quick reaction in real-time. The second loop proceeds concurrently to the first one and keeps track of history data. For every action at time t, the situation at $t + \Delta t$ is measured by the observer and written to a memory. We use these tuples of actions and resulting situation parameters for evaluation and calculation of new fitness values (F_i), which are updated in the mapping. In order to avoid overcontrol we define a fixed time delay Δt to assign results to preceding actions.
The generic architecture does not specify the mechanisms of machine learning in detail. Possible methods could be artificial neural networks, learning classifier systems, reinforcement learning or evolutionary algorithms. Adaptation can occur online and/or by a model-based internal learning process, which is introduced by the simulation model in combination with the adaptation module in a third control loop. OC scenarios are mostly real-time systems with hard time restrictions. Thus, offline learning is often not feasible, but e. g. evolutionary algorithms could be used for generating completely new rules, and for modification of existing rules with genetic operators (mutation and crossover). Simulation would be needed to predict the success of control actions.
Below, the different decisions are listed, which have to be taken for every control loop. The workflow may differ with respect to the scenarios and may depend on implemented mechanisms of machine learning.

- Is there an action of the controller expected? Is there some kind of unanticipated behaviour observed? Are there some indicators exceeding their predefined thresholds? The decisions are based on local reference values for the indicators and degradation of the objective function.
- In order to avoid repeated action and overshooting control, the system has to remember recent responses to a situation and to wait some time Δt for the effect to show.
- What is the best action for the observed situation parameters? The mapping is browsed for a best matching tupel consisting of condition (C_i) and action (A_i).
- If no suitable control rule exists in the mapping, the adaptation module has to generate a new one. If the controller has access to a system model, it can assess such new control actions in a simulation model before actually applying them to the real system. If time permits, such a model would allow an internal response-optimisation.
- What is the reward for the action responsible for the present situation(s)? The corresponding fitness value is changed depending on the success of the rule.

5 Conclusion and Outlook

The journey to get closer to the overarching vision of OC has just started. Our ideas will involve an evolution of innovation in systems and software engineering as well as collaboration with many other diverse fields. In this paper we discussed an organic architecture

consisting of two parts – an observer and a controller – and presented OC as a vision based on the urgent necessity to find methodologies for managing the complexity and controlling the behaviour of large scale distributed embedded systems. It was shown how observer and controller are designed, which functions should be implemented and how the loop consisting of the SuOC, an observer (observing the behaviour of the SuOC in terms of well defined system parameters) and a controller (selecting adequate actions to optimise system behaviour with respect to certain global objectives) work together.
The components of the OC architecture strongly relies on other established scientific areas like data mining, time series analysis, machine learning, or control theory. Results and methods from these areas will be used to extend the observer toolbox and to test our metrics and control strategies with different scenarios. Currently, our concepts are tested with respect to some multi-agent like scenarios (a multi-elevator system and a collection of freely moving simple robots showing some undesired emerging effects). As a next step we plan to use our framework in a practically usable traffic control system with local and global observer/controller components.

Acknowledgment: This work has been done within the DFG Priority Program 1183 Organic Computing [DFG05]. We are especially indebted to Fabian Rochner, Universität Hannover, and Holger Prothmann, Universität Karlsruhe (TH), for their valuable suggestions.

References

[DFG05] DFG Priority Program 1183 Organic Computing. `http://www.organic-computing.de/SPP`, 2005. visited May 2006.

[KC03] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 1:41–50, 2003.

[MMS06] 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.

[MS04] 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*, pages 2–5, 2004.

[MSvdMW04] C. Müller-Schloer, C. von der Malsburg, and R. P. Würtz. Aktuelles Schlagwort Organic Computing. *Informatik Spektrum*, 27(4):332–336, 2004.

[Sch05] H. Schmeck. Organic computing: A new vision for distributed embedded systems. In *Proceedings Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005), 18-20 May 2005, Seattle, WA, USA*, pages 201–203. IEEE, IEEE Computer Society 2005, May 2005.

[SMS05] T. Schöler and C. Müller-Schloer. An observer/controller architecture for adaptive reconfigurable stacks. In M. Beigl and P. Lukowicz, editors, *Systems aspects in organic and pervasive computing – ARCS 2005*, pages 139–153, March 2005.

[Ste05] R. Sterritt. Autonomic computing. *Innovations in systems and software engineering*, 1(1):79–88, March 2005.

[VDE03] VDE/ITG/GI. Positionspapier Organic Computing. `http://www.gi-ev.de/fileadmin/redaktion/Presse/VDE-ITG-GI-Positionspapier_20Organic_20Computing.pdf`, 2003. visited May 2006.