

## Applications and architectures in organic computing

**Matthias Güdemann, Florian Nafz, Andreas Pietzowski, Wolfgang Reif, Benjamin Satzger, Hella Seebach, Theo Ungerer**

### Angaben zur Veröffentlichung / Publication details:

Güdemann, Matthias, Florian Nafz, Andreas Pietzowski, Wolfgang Reif, Benjamin Satzger, Hella Seebach, and Theo Ungerer. 2006. "Applications and architectures in organic computing." Augsburg: Universität Augsburg.

### Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under the following conditions:

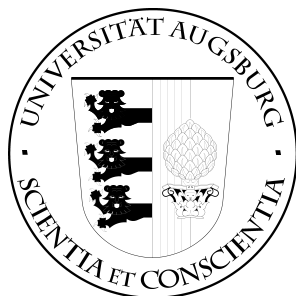
**Deutsches Urheberrecht**

Weitere Informationen finden Sie unter: / For more information see:

<https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publizieren/>



UNIVERSITÄT AUGSBURG



**Applications and Architectures in  
Organic Computing**

(Priority Program "Organic Computing" Nr. 1183 of the DFG)

**M. Güdemann, F. Nafz, A. Pietzowski, W. Reif,  
B. Satzger, H. Seebach, T. Ungerer**

Report 2006-21

October 2006

**INSTITUT FÜR INFORMATIK**  
D-86135 AUGSBURG

Copyright © M. Güdemann, F. Nafz, A. Pietzowski, W. Reif, B. Satzger, H. Seebach, T. Ungerer  
Institut für Informatik  
Universität Augsburg  
D-86135 Augsburg, Germany  
<http://www.Informatik.Uni-Augsburg.DE>  
— all rights reserved —

---

## **Abstract**

This technical report documents a summary of the SPP-OC Workshop "Architectures and Applications" at the University of Augsburg. Architectures and applications of organic computing systems are two of the profile topics of the priority program "Organic Computing" of the German Research Foundation (DFG SPP 1183). The report contains the discussion results to the several questions discussed on the mentioned workshop.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Questions</b>	<b>3</b>
2.1	OC-aspects, OC characteristics and OC-feeling . . . . .	3
2.1.1	How are the OC-aspects generated? . . . . .	3
2.1.2	Should there be a "OC-feeling"? . . . . .	4
2.2	OC-Design and surplus . . . . .	5
2.2.1	Contradiction between top-down and non-controllability? . . . . .	5
2.2.2	Differences between OC and non-OC implementations? . . . . .	5
2.3	Connection between Architecture and Application . . . . .	6
2.3.1	Where are the organic computing aspects generated? . . . . .	6
2.3.2	What demands does the application impose on the middleware and vice versa? . . . . .	6
2.4	Clarification of the different perceptions of architecture . . . . .	7
2.5	Principles and differences of OC techniques . . . . .	8
2.5.1	Which principles are used in OC architectures? . . . . .	8
2.5.2	What is the difference between OC architectures and non-OC architectures? . . . . .	9
2.5.3	What is the benefit of OC architectures? . . . . .	9
2.5.4	How can the benefit be measured? . . . . .	9
<b>3</b>	<b>Conclusion</b>	<b>10</b>

## 1 Introduction

Rising complexity and requirements claim new types of systems and other methods for development. These new systems should be intelligent, very flexible, robust, safe and adapt to human needs. They should control the complexity and could reach aims on their own without giving explicitly every command. Organic Computing based on the insight that we will have more and more autonomous systems, which have sensors and actuators to observe their environment and perform actions according to. An Organic Computing system is a system, which adapts dynamically to its environment and its conditions. Therefore, such a system will have several self-x properties, like self-organizing, self-healing, self-protecting, self-reconfiguring and context-awareness. The priority program "Organic Computing" of the German Research Foundation (DFG) deals specifically with this issues. The profile topics of this program are:

- Applications
- Architectures

- Design methodology
- Large networks
- Nature and technique
- Self-organisation and emergence

This technical report represents a summary of the documents the SPP-OC Workshop "Architectures and Applications" on the first two profile topics applications and architectures at the University of Augsburg from 20th till 21st of July. At this workshop several topics and questions on these two profile topics were discussed. In the following we give a summary of the discussions to the respective questions and their results.

## 2 Questions

In this section we want to summarize the discussed questions and issues. To each point we give a short description of the arguments and the discussion results.

### 2.1 OC-aspects, OC characteristics and OC-feeling

One discussion topic of this workshop was "OC-aspects, OC characteristics and OC-feeling". This topic covers the questions, if there should be a difference in using an OC system and a non-OC system or not, what characterizes an OC system and how are these characteristics generated.

#### 2.1.1 How are the OC-aspects generated?

Organic Computing aspects are the properties of a system that are characteristic for an "organic" system. OC-aspects are for example the self-x properties. Some of the OC-aspects of the system are generated in the application, middleware or hardware and some are generated by cooperation. A middleware, for example could provide services which are used by the application to generate self-x properties. For the property generation conventional techniques plus combination techniques for generation of the self-x properties are used.

There are a lot of different methods which are an encouragement for the generation methods of self-x properties, like hormones, ants and swarms. Some methods how OC-aspects are generated are:

- Redundancy, dynamic redundancy
- Miscellaneous learning techniques
- Communication between components

Of course there are more than these three mentioned methods, but these were the most designated ones. All in all, OC-aspects are generated by different methods and in different ways.

Another arising questions is: "When are the OC-aspects generated?" Are they generated during design time, the initialization of the system or during runtime? One answer of this question is, that there is not one point of time that is the right one. It is level constrained. On the hardware level most aspects have to be fixed and generated during design time or initialization. For example, it is hard to change a conductor during runtime. While on application level the generation during runtime is conceivable.

### 2.1.2 Should there be a "OC-feeling"?

The term "OC-feeling" means, that the administrator, developer or user recognize that the system he deals with is a Organic Computing system. The user should recognize a difference between a ordinary system and an organic system - directly or indirectly. An organic system can stand up to an ordinary system and it has to be better. "Better" in the sense of more failure tolerant, more comfortable, easier to use or has properties ordinary systems do not have. An organic system should seem for the user to be intelligent. For example, the user recognizes this through more information like error explanation or status information. The use of an organic system should be easier because the user just has to give aims but not the way how to reach them. The development of organic systems should also be simpler, because not everything has to be coded. The user should notice a positive intelligence of the system. Positive means that the intelligence is in the interest of the user. For the user the OC-feeling of an Organic Computing system consists of:

- higher reliability
- easier handling
- softness of the system
- graceful degradation

In this context softness means that the system is not fixed, but it is more flexible and it works even if some functions are broken. Graceful degradation is to sustain the base functionality as long as possible. Even if some failures occur the system restricts its functionality as much as needed, but tries to fulfill as much functions as possible. For the developer or administrator the following points should be characteristics for an OC system:

- less effort for development
- no or minimum effort in case of a failure
- less administration effort

In summary, the user, administrator and developer recognize an "organic" system by its easier handling and greater flexibility compared to conventional systems.

## 2.2 OC-Design and surplus

Traditional systems could be designed in a normal top-down approach. Organic systems show such a complexity that the normal way of designing computer systems is no longer possible. Therefore this discussion session had the focus on new ways and possibilities for designing OC systems.

### 2.2.1 Contradiction between top-down and non-controllability?

To a certain extent top-down design is possible. In systems which are ad-hoc and massively distributed it is difficult to design the whole system in a top-down way. The several components could be designed top-down whereas the interaction between these components must probably be designed bottom-up. Therefore the use of genetic or evolutionary algorithms could be used. There arises a new question: Do we use evolutionary algorithms to design and develop the communication algorithm or do we implement an evolutionary algorithm as communication algorithm? Both ways are conceivable.

Additionally the observation was discussed, that behaviour which is already observable could be designed in a bottom-up way. Therefore an approved method could or should be consulted. Some methods which were mentioned are:

- design by hand
- evolutionary algorithm (with online or offline simulation)
- genetic algorithm
- nature inspiration (e.g. ant algorithm)
- construction principles

Concluding we can say that there exists no serious contradiction. But at the moment there is no well-known way to design the whole organic system in a top-down way.

### 2.2.2 Differences between OC and non-OC implementations?

The main difference between these implementations is that Organic Computing implementations could handle faults, failures and changing circumstances. At an OC system failures are tolerated and the challenge is to get them under control. Failures have to be controlled within a given scope. They must not lead to a disaster or the system must be able to repair them without a hazard occurring. These organic systems could fix faults and failures or adapt to new circumstances. Organic Computing systems have sensors to recognize a changing environment and actuators for performing resulting actions. These systems do react on their own on changing environment or requirements. We say, they have self-x properties. For example self-healing, the systems repair itself if some functionality is broken. Or self-adaption, the system change its behaviour according to changing environment or aims. At an OC system not every behaviour

is coded which leads to that not every behaviour has to be considered in the development process.

A second difference is the approach which is used to design such organic systems. Most of them are "bio"- or "nature"-inspired. The designers analyse natural behaviour and try to transfer this on computer systems. At this the challenge is the transfer of these behaviours to a technical system. An example for this task are ant algorithms or swarm behaviours.

So the differences between OC and non-OC implementations are in the approach, the properties and the handling of failures. The OC approach is, as the name implies nature inspired. OC systems have several self-x properties and act as far as possible autonomous. And third, OC systems could handle occurring failures in a given scope.

## **2.3 Connection between Architecture and Application**

Many of the participants of the workshop showed interest in both topics, architecture and applications. Therefore one discussion topic was targeted at the connection between these two topics.

### **2.3.1 Where are the organic computing aspects generated?**

When designing an OC application, many choose a layered architecture with different aspects addressed in each layer. This can for example be a simple architecture with a hardware layer, a software layer and an additional middleware layer. Other layered variants exist, some only seen from different viewpoints.

The self-x properties and more general OC aspects can be generated in any of the layers, but with different degrees and with different metrics. In the example above the hardware has very limited self-x properties, the software has some application specific ones and the middleware is responsible for non-functional qualities like self-optimization. The degree of self-x capabilities will always depend on how it is realized and on the demands on the layer or part where it is implemented. Real-time hardware or software will allow only for a limited degree as response times must be maintained. A system layer that has no real-time requirements provides more opportunities for broader self-x mechanisms. The layers themselves can recursively be separated in layers where the same classification may be applicable.

When accepting that OC properties can be generated in any layer of the system, it is possible to build both an organic application with a conventional middleware and to make a system organic by using the principles in the middleware layer.

### **2.3.2 What demands does the application impose on the middleware and vice versa?**

Like in conventional non-organic systems, a middleware provides services for the application. For many of the organic applications, these services incorporate at least a communication method, as most are distributed architectures. It was found that there are applications that consist to a larger degree of a middleware

than others. This is especially true for sensor networks where information exchange by communication is the essence of the application. Other application impose less demands on the middleware and may even be operative without any middleware or be able to tolerate a failing middleware layer.

To design a organic middleware that is not reduced as communication service, it is essential for the application to provide a feedback system. Application specific information must be provided by the application itself. If the middleware serves as observer then it must be capable of noticing the need for a change in the application, but which change or if a given change is valid must be decided by the application itself.

This characterizes the separator between the application and the middleware. A load balancing middleware is a very general concept that can be used to generate many different self-x properties. The above mentioned separator gives a hint why not all problems can be solved as load-balancing instances.

## **2.4 Clarification of the different perceptions of architecture**

The purpose of this discussion was to acquire consistent notions of the term “architecture” and to identify categories that are adequate to describe the different notions of architectures of the projects of the priority program.

Several categories have been identified in the run-up to the workshop:

- type/goal of project
- type of architecture
- type of Observer/Controller (O/C) architecture
  - O/C levels
  - central/distributed O/Cs
  - hierarchically nested O/Cs on a node
  - O/Cs in application system integrated or independent
- hardware requirements
  - node capability (PC, PDA, ...)
  - typical number of nodes
  - energy restriction
  - communication protocol
  - communication topology
- system requirements and targets
  - self-x capabilities yielded
  - soft/hard realtime requirements
  - types of learning techniques applied

These listed categories have been subject to enhancements and extensions:

An interesting aspect to analyse the architecture of an OC system is to check whether the observer/controller of the system is integrated into the application or independently usable. Also the organic concepts can be classified whether they are application specific or general-purpose. This offers valuable clues to the used and developed concepts.

Very helpful insights into the architecture of a project is provided by the information what architectural structures or methods enable the self-x properties and learning capabilities of a system. Also the knowledge how organic computing systems exploit redundancies would grant better understanding. All these information could help to figure out the concepts, benefits, and functionality of OC systems in a better way.

To prevent misunderstanding it is useful to introduce a category that reveals whether the architecture of a project corresponds to the field of hardware, middleware or application.

To classify the architectures and to establish a border between OC and other systems it is helpful to point out to what extent techniques of the observer/controller go beyond control theory techniques.

It has been stated that the categorization of the architectures of the projects of the priority program should focus on the special aspects of organic computing systems to identify generic concepts of the used architectures.

Finally, a further result of the discussion has been the conclusion that the an observer/controller can also exist in applications.

## **2.5 Principles of the OC techniques used in the different architectures**

Different architectures use different approaches and techniques to implement their specific organic features. In this part of the workshop the discussion should clarify which principles were used and if there are similarities between the projects.

### **2.5.1 Which principles are used in OC architectures?**

Most OC architectures take a leaf out of nature's book. The principle of redundancy is very widespread in nearly every biological system. It is also typical for living organisms that a component is not responsible for only one specific task or function but can handle different jobs. This results in high reliability when one component fails because others can help out in that case. A design with purely centralized components is neither useful for nature nor for a robust OC architecture. Only in some cases or specific application areas a centralized solution should be considered. Another principle and a similarity of nature and OC architectures is that local interaction between components may result in global behavior. In difference to the evolutionary nature the human built OC architectures are often designed top-down.

**2.5.2 What is the difference between OC architectures and non-OC architectures?**

Conventional architectures surely can solve many problems but potentially fail to cope with all possible cases that can appear in a system. An OC architecture should be designed with plasticity. That means that such an architecture should have some degree of freedom to adjust itself to the problem in contrast to a conventional design which constricts the search space. Learning capabilities are one way to reach that. The main goal is that OC systems can manage sudden phenomenas and cover unexpected situations. The user should use the system as a whole without need to be familiar with the components inside. Controlling the system should be done rather with declarative programming than with procedural programming. Another difference is that OC architectures are aware of taking advantage of emergent effects and may contain new algorithms and network protocols to be self-adjustable.

**2.5.3 What is the benefit of OC architectures?**

The main benefits of OC architectures can be summarized in a few buzzwords: adaptability, robustness, flexibility, learning aptitude, resilience, reliability, effectiveness and efficiency. In the best case the system gets improved in all these aspects using an OC architecture. For example the overall life time of an energy restricted sensor network should be increased or the load should be balanced between components. A system should be more robust in more dimensions if unforeseen circumstances arise. Conventional architectures mostly optimize robustness in only one single dimension (e.g. speed at the expense of quality). As systems will get more and more complex in the future another benefit of OC architectures is the controllability of that complexity. The reduction of external complexity is reached by (temporarily) increased internal complexity. Furthermore it will be even possible to meet certain challenges which are not meetable with conventional techniques. This also results in shorter cycles of development what exculpates the software developer.

**2.5.4 How can the benefit be measured?**

To measure the improvements of architectures using OC techniques statistics are used very often. This is only practicable when all acceptable system states are known or at least a model of failures and a metric of quality exist. The model of failures is divided into an inner and an outer model related to the architecture. The metric of quality is always coupled with the application and thus can be very different in some cases. Basic improvements like reliability, speed-up, or the time needed for developing a system are simple to measure. If no real data is available the measurement results can always be estimated with a worst-case analysis.

### 3 Conclusion

The miniworkshop for the two profile topics "Applications and Architectures" of the DFP-SPP "Organic Computing" was a success. New ideas were found for the selected topics in the discussion groups and a compendium of the available applications and architectures was made. The five thematic topics were discussed in detail during this workshop with some satisfactory results, as you can see in this report. Thanks to all participants on these prosperous discussions.