

Order and C⁰h_as

An overview of Organic Computing

Oliver Gableske
For the Institute of Distributed Systems
University of Ulm
Ulm, Germany
oliver.gableske@uni-ulm.de

ABSTRACT

This paper is meant as an introductory work on the research field of Organic Computing (OC). It gives information on the main goals of Organic Computing and provide the basic ideas behind it. Furthermore, it explains the basic terminology that is required in the research field of OC and give some examples for recent work. The conclusion gives a brief overview on how the position in OC is today. Due to the fact that this is an introductory paper, it will contain numerous references for supplemental reading.

1. INTRODUCTION

The past two decades have been characterized by the development of complex (distributed) systems [16]. These systems continue to merge into the society and make an anticipatory research necessary [1]. They reach from Personal Area Networks (PANs), as they are often referred to in Ubiquitous Computing (UC), over automotive systems, and factory automation, to the widely known telecommunication systems like GSM and the Internet. More complex systems provide more features and can complete more complex tasks, but are more difficult to maintain and manage at the same time. Due to the fact that we become more and more dependent on these systems, they ought to be constructed in a trustworthy, safe, robust, and flexible way [16]. These complex systems have to be different from those that are used nowadays in order to stay manageable by human beings. If we cannot reduce the administration efforts to manage these systems, we will run out of skilled I/T personnel to keep the new systems up and running [11]. Therefore, these systems have to manage parts of their existence by themselves, e.g. have to be self-organizing, self-healing, self-adapting, self-protecting etc. Furthermore, they have to be context aware and must emerge important structures by themselves. The construction of systems providing these properties is one of the goals in OC. All these properties of a so called Organic Computer System (OCS) will make sure, that these systems can be used widely as it is foreseen for Per-

vasive and Ubiquitous Computing, MANETs, and systems where manual management is too difficult, too expensive, or too time-consuming [7]. OCSs are built with silicon technology and do not integrate biological components. They do not use chemical reactions or other physical effects to compute than those already known to today's IT personnel. It is therefore not necessary, that management personnel must be trained in fields of biology, chemistry or physics to handle OCSs. They can be implemented on nowadays hardware, thus the word "organic" does not refer to biological parts. These systems are called organic, because their paradigm lies in natural phenomena, mostly found in living beings (like ant-colonies, bird-swarms) or physical effects (like clustering of atoms). There are technological approaches as well as approaches brought forward by nature science to reach the goals of OC (see Figure 1) [17]. However, as it is stated in [11], the success of an OCS (as it is with all important software systems), depends on their systematical software engineering. Without the proper methodologies, it is not possible to cope with the complexities of life-like systems, especially with their self-x properties [11].

In summary, Organic Computing tries to use natural phenomena as a paradigm, thereby adapting concepts of biological systems in silicon technology and software, that can be seen as life-like. These efforts are undertaken to develop architectures that can be used in the creation of complex distributed systems, and to make future systems more manageable by redirecting some of the administrative effort back to the system itself.

Before the complexities behind systematical software engineering in OC can be understood, it is necessary to first understand the basic terminology that is used in this field of research. The following section deals with this terminology and was mostly inspired by [25].

2. SELF-X PROPERTIES VERSUS EMERGENCE

This chapter focuses on the terminology of Organic Computing. Two major classes of terms have developed so far: *self-x properties* and *emergence*. Since OC is a new field of research, some of the terms described and defined here are still a debated issue [25]. The following sections show why researchers have a hard time to define the terms mentioned above.

2.1 The self-x Properties

The term *self-x* itself refers to several properties that can

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Seminar FVS 2007

Copyright 2007 Oliver Gableske.

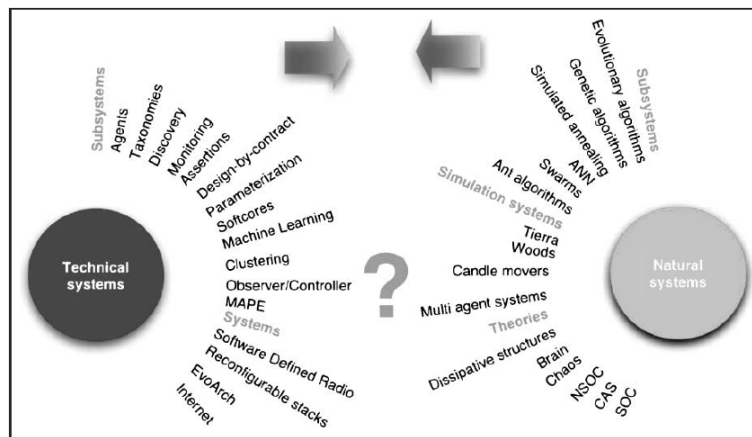


Figure 1: Approaching the goals of organic computing [17].

be observed in natural systems, for example:

- self-organizing
- self-healing
- self-protecting
- ...

This section will focus on self-organization, since it is the central property of an application that is considered “intelligent” or “life-like”. A systems that contains self-protection will be described exemplarily in Section 4.3.

2.1.1 Historic Overview

The concept of self-organization is very old [25]. René Descartes wrote about self-organization back in 1637, using phrases like “arrange itself”¹. Although the concept of self-organization is quite old, the term as we use it today has only been used since the 40s [25], firstly appearing in a paper by W. Ross Ashby [2]. Unlike most researchers today, Ashby seemed to have a very clear understanding on what self-organization is:

Definition 1. (Organization, Ashby)

The organization of a system is the functional dependence of its future state on its present state and its external inputs, if any.

That means, that Ashby understood a system to be self-organizing, if it changes its own organization, thus changing its own future state. This definition explicitly allows a system to use external input even if the input is generated by the system itself.

Several more research domains studied self-organization [25]. Among these were physics, systems theory and computer science, and self-organization is nowadays used in many disciplines. In computer science, self-organization is often mentioned in the area of multi agent systems (MAS). For supplemental reading see [18]. This work contains information on cooperation and group formations, as well as some examples for realizations of self-organization, namely in networks, robotics and vocabularies between agents.

¹The mentioned work is called “Discours de la méthode pour bien conduire sa raison et chercher la vérité dans les sciences”

2.1.2 A modern definition for self-organization

De Wolf and Holvoet tried to invent a definition for self-organization that is supposed to be consistent with the historic understanding of it, hence it had to include the ideas of “increase in structure” and “autonomy”, since these concepts are included in the terms “self” and “organization” already [25]. Furthermore, De Wolf and Holvoet added the concepts of “Adaptability or Robustness w.r.t. changes” and “far-from-equilibrium” to propose a modern definition:

Definition 2. (Self-Organization, De Wolfe and Holvoet)

Self-organization is a dynamical and adaptive process where systems acquire and maintain structure themselves, without external control.

Throughout the rest of this document, self-organization will be understood according to Definition 2, but as stated initially, the continuing dialogue between scientists may revise the understanding of the term. The ideas that are included in Definition 2 must be characterized in order to understand the difference between self-organization and emergence.

2.1.3 Concepts in self-organization

Definition 2 contains a few concepts that must be understood, before taking a look at the term of emergence and to be able to tell the difference between emergence and self-organization. [25] gives a more detailed overview on the characteristics that follow.

Increase in Order: Increasing order refers to a state change of a system, where its components arrange in a certain way, so that the system as a whole can promote a specific function. It is not possible for a system to fulfill a certain function without a certain arrangement of its components. This arrangement is called order as soon as it promotes this function. Without order, the function cannot be fulfilled and without a function, there cannot be any order, because the assessment criterion is missing. This indicates, that a system can not only be too unordered, but it can also be in too much order, hindering it from fulfilling a certain function. Thus, systems must find a balance between no order and too much order.

Autonomy: The second important concept that lies within

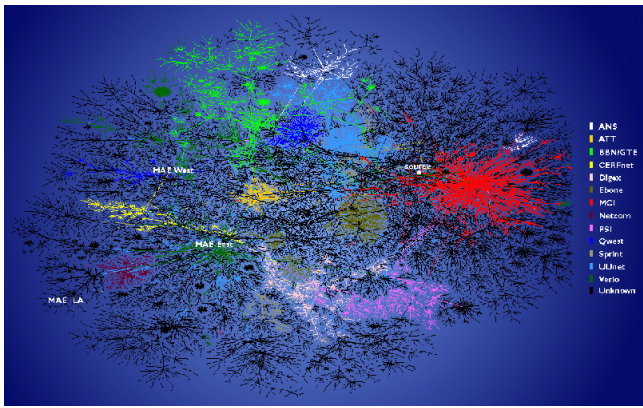


Figure 2: Self organizing patterns: Internet topology [6].

Definition 2 is, that the proposed state change is done autonomously and without any external influence. This does not mean, that the system must not use any collected data or cannot recycle its own calculations. It means, that the decisions on what to do with this data must be made within the system itself (e.g. no controlling instance). [25] gives the example of plugging in a USB device. The plug-in process is triggered by an external entity (the user) but the decision on what to do with the information/device and how to configure it is made only within the machine. The machine handles the situation autonomously.

Adaptability vs. Robustness: A self-organizing system is supposed to cope with changes in its environment and is therefore considered as an adaptable system. An opposing force to adaptability is robustness, e.g. keeping a certain order without respecting changes in the environment. Having an adaptable system (that is not robust) does not only mean, that it is most flexible and can adopt to any new situation, it also means, that it becomes uncontrollable. On the other hand, having a system that is too robust means, it can easily be controlled but lacks flexibility, and thus is not self-organizing anymore. It becomes clear, that the system must be confined to a certain level of adaptability. This ensures, that the system is robust and controllable. At this point, it is necessary to introduce one more term:

Definition 3. (Attractor)

An Attractor is the set of states, that an adaptable system is confined to.

This means, that by committing a self-organizing system to a certain (well defined) attractor, you set the horizon of the system, in which it will be able to adopt and re-organize itself. There are a few attractors mentioned by [25], e.g. a point-attractor (leaving no room for re-organization), a limited cycle-attractor (that allows a finite adoption and periodic behaviour) and a chaotic attractor (that allows a huge amount of behaviours).

Far-from-equilibrium: Self-organization is a process in time and thus, has a certain dynamic. A system that is far-from-equilibrium is more dynamic, e.g. changes faster, than a system that has reached an equilibrium. Environmental changes that appear around a far-from-equilibrium



Figure 3: Self organizing patterns: The Belousov-Zhabotinsky reaction [16].

system will affect this system much more, than a system, that is balanced out. Contrariwise, a system that is far-from-equilibrium can handle changes much faster, e.g. will self-organize faster. There is a mathematical approach in [20] that deduces the ‘far-from-equilibrium’-state as a necessary prerequisite for self-organizing systems.

Simple examples of self-organization are presented in Figures 2 and 3.

2.2 Emergence

Emergence is frequently confused with self-organization [25]. This is due to the fact, that people often describe ‘emergence’ as some kind of global system behaviour, that arises out of the systems components by interaction. As this is also a property of self-organization, it is easy to mistake one for the other. To clarify the understanding of emergence, a historic overview is helpful and will be given in the next section.

2.2.1 Historic Overview

The roots for emergence as a term are, like self-organization, quite old. Some of the basic concepts that are included in emergence can be found in teachings of the ancient Greeks and later on as ideas in western societies. Two of the basic ideas that influenced the modern understanding of the word are: ‘the whole before its parts’ and ‘Gestalt’ [25]. Whereas ‘Gestalt’² refers to the unreducibility of the system as a whole, meaning that one cannot understand the behaviour of the system when only studying its parts. The phrase ‘the whole before its parts’ refers to predominance of the system behaviour before the behaviour of single components, thus making it more important to understand the system’s global behaviour and not the behaviour of its local components.

The first time when ‘emergence’ was mentioned seems to be in 1875 when the philosopher Lewes [12] differed between ‘resultant’ and ‘emergent’ chemical compounds that came about from a chemical reaction [25]. Lewes proposed, that a chemical reaction is to be called ‘emergent’, if the steps of the chemical process could not be traced back in order to see how much share each component had in its resultant.

²Gestalt (ger.) means shape or form. However, there is no accurate translation for this word, because it means not only the visible parts, but it also refers to the design and substance of something.

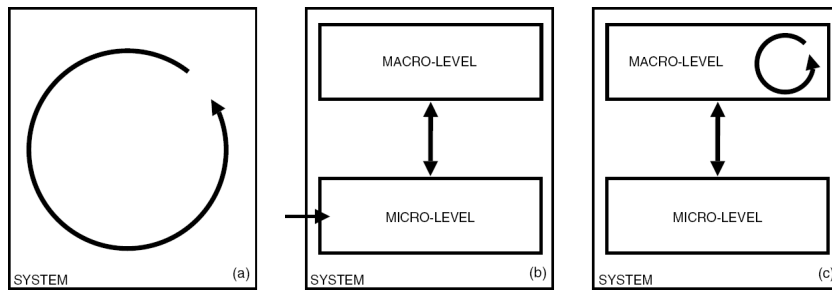


Figure 4: Emergence and self-organization, combined [25].

The term of ‘emergence’ was later on studied and borrowed by many theories, namely

- Complex adaptive systems theory
- Nonlinear dynamical systems theory and Chaos Theory
- The synergetics school
- Far-from-equilibrium thermodynamics

and others. For a more detailed overview of the history of emergence see [10].

2.2.2 A modern definition for emergence

As De Wolf and Holvoet did with self-organization, they proposed a definition for the term emergence w.r.t. its historical understanding [25].

Definition 4. (Emergence, De Wolfe and Holvoet)

A system exhibits emergence when there are coherent emergents at the macro-level that dynamically arise from the interactions between the parts at the micro-level. Such emergents are novel with respect to the individual parts of the system.

The word ‘emergent’ refers to the result of the process of emergence. These emergents can be properties, behaviours, structures, patterns, etc. The micro/macro-levels refer to a point of view, namely the system as a whole (macro-level) and the sole components of the system (micro-level). To fully understand the definition for emergence, one has to know the concepts that influenced it. These concepts have evolved over time and help to understand the word ‘emergence’. Therefore, the following section will explain them in more detail.

2.2.3 Concepts in emergence

Definition 4 contains several aspects that must be understood, before any comparison between self-organization and emergence can be done.

Micro-Macro effect: Concerning to [25], this is the most important characteristic of emergence and it is stated in most of the literature. A micro-macro effect is given, when interactions on the micro-level create properties on the macro-level. These properties can be patterns, structures, behaviours, etc. These properties are called ‘emergents’, thus emergents are results from the micro-macro effect.

Radical novelty: The radical novelty refers to properties on the macro-level, that cannot be linked to any entity in

the micro-level. Usually, properties that are that novel do not make any sense in the context of the micro-level. For example water-molecules align themselves in clusters, thus clusterize would be an emergent of water, but it makes no sense for a single water molecule. Furthermore, it is not possible to deduce the macro-level properties from its micro-level properties. In addition to the water-example before, one cannot predict the ‘Gestalt’ of the clusters that will emerge, though he knows exactly how many water he might use in the experiment of clustering water. But caution is advised: stating that macro-level properties cannot be reflected by micro-level entities is a serious misunderstanding. The effect of emergence means, that the macro-level property can not be deduced from its micro-level entities, due to too high complexity. However, the emergent might be studied in context of many (not all) micro-level entities.

Decentralized: Decentralization means, that there is no central entity (whether it is internal or external to the system), that directs the macro-level behaviour.

Two-Way link: In addition to the micro-macro effect, the two-way link encases the existence of feedback from the macro-level behaviour back to the micro-level entities, thus micro-level entities which (however) form the macro-level property, will indirectly be affected by this property and thereby affect themselves. See Figure 4b.

Robustness and flexibility: One of the most eligible effects of emergents is, that they are robust and flexible at the same time. No single micro-level entity holds all the information that let the macro-level property emerge. Hence, the failure of one single micro-level entity will not affect the macro-level structure in a dramatic way. Due to the decentralization, it takes the malfunction of many micro-level entities to affect the macro-level property in a noticeable way. In other words: the quality of the output will decrease gradually, without sudden loss of performance [25]. The flexibility of the system can be explained similarly. Reconfiguration of the micro-level entities will push the system slowly towards a new (better performing) configuration, though the effect on some micro-level entities may be dramatically.

2.3 Comparison and Combination

Self-organization and emergence can appear independently. Reduced to its core, self-organization is an adaptable behaviour that creates and maintains order. Emergence in its core is the existence of novel properties, that can not be reflected by the system’s sole entities. The similarities between both are given in [25]. Self-organization and emergence are

- dynamic processes arising over time.
- robust. Whereas the robustness in self-organisation refers to its flexibility to adopt and thereby maintain order. The robustness of emergence refers to its graceful degradation of output-quality (no immediate system failure).

The differences between self-organization and emergence have been presented through the different concepts that are embedded in the terms (see section 2.1.3 and section 2.2.3). Though emergence and self-organization are not prerequisite to each other, it seems to be a promising approach to combine both in the way [25] proposes (see Figure 4). There are two main approaches that can be used for combination. On one hand, self-organization could be the basis that influences the micro-level which is thereby incited to promote a certain emergent effect (see Figure 4a and 4b). On the other hand, self-organization could be an emergent (thus, the self-organizing feature of the system is a property that arises out of emergence). This is visualized in Figure 4c. Supplemental reading on the topic of emergence or self-organization, examples and additional propositions of definitions can be found in [25, 15, 22, 7, 8].

3. THE ANTAGONISM OF CONTROLLED EMERGENCE

As stated in the introduction, the success of a computer system is depending on its systematical software engineering. Therefore, it is necessary to embed the concepts, that have been discussed in Section 2, into a software engineering process. As we know from the concepts that are embedded in the term of emergence (Section 2.2.3), emergents, that evolve dynamically and without centralized control over the micro-macro link, cannot be explained in a deductive way (due to complexity). The main problem is the micro-macro link, because it cannot be understood and makes emergents look “magic”. Moreover, working on the micro-macro link is not enough, because it does not contain the feedback from the macro-layer to the micro-layer, thus mastering the two-way link would be more useful. However, it is not easily possible to design OC software in a rigid scheme (like the classic waterfall-model). De Wolf and Holvoet propose, that a promising approach to solve this problem lies within the combination of self-organization and emergence as it was described in Section 2.3, but they leave the question on how this can be achieved unanswered. As Fromm describes in [7], it is necessary to answer the following questions:

How can a globally desired structure or functionality be designed on the basis of interaction between many simple modules? Can we solve the micro-macro link problem?

Due to Fromm, this is the most central question for these days research in the field of OC. Methodologies must be found, that can involve emergence into today's (then organic) software by making emergents manageable in (organic) software engineering systems. As described in [7], the most promising approach to achieve this goal seems to be the modification or reinvention of the engineering processes for Multi-Agent Systems (MASs). Several architectures and methodologies have been developed so far, among them are

MASSIVE (Multi Agent Systems Iterative View Engineering), which is a German development by Jürgen Lind [13] and

ADELFE (Atelier de Développement de Logiciels à Fonctionnalité Emergente), which is a French development by Bernon and Gleizes [4].

These two are considered the most interesting architectures by [7], since they support emergent behaviour (ADELFE) and emphasize the importance of “Round-trip Engineering” as well as “Iterative Enhancement”. These features are supposed to solve the problem of the micro-macro link. As mentioned in the introduction to this section, the development of (organic) software engineering systems is still a matter of actual research and will therefore not be discussed here. The interested reader can find additional information in [7, 13, 4, 11, 3].

4. EXAMPLES FOR ORGANIC COMPUTING

After explaining the basic terminology in OC and presenting some of the actual research problems in the last sections, the following section will focus on examples of Organic Computing. The examples given below are far from being applicable software tools. They are a proof of concept for some aspects in OC and provide vivid examples for the concepts of self-organization and emergence. Furthermore, the examples given here are just outlined and the interested reader can find further information in the respective papers. In addition, there are more complex applications that contain the ideas of OC that can be found in [5, 11, 21, 24].

4.1 The AMIDAR Class of Processor

Gatzka and Hochberger developed an architecture for a new type of processor class: The AMIDAR (Adaptive Microinstruction Driven Architecture) [9]. They propose, that Configurable Systems on a Chip (CSoC) are becoming more and more important in the embedded systems market. CSoC are a result of the improvements in microelectronics, and their availability makes it possible to reconfigure the chip while in use. AMIDAR will disclose the possibility of processors that (self)-organize their overall structure to better satisfy the needs of the application currently occupying the processor. AMIDAR is no hardware processor yet, but it was simulated by the derivation of a Java bytecode processor. More information about these simulations follow at the end of this section.

The fundamental architecture of the processor is given in Figure 5a. It consists of functional units (FU), a token generator, the token distribution system and a communication structure. FUs are a piece of hardware that execute specific tasks on the processor. They are thereby controlled by tokens. In other words: the FUs do the actual work, concerning to the content of the tokens. The tokens used here are not only an instrument to limit the bandwidth that FUs use up on the communication network, they also provide information on where FUs are to send operation results to. Although the functioning of the processor and the way it reorganizes itself is too complex to be explained here in detail, the basic principle on how it works is simple: The configuration manager (Figure 5b) collects statistical data on the FUs and the bus systems (token distribution and communication structure). It then decides whether any adaptation is necessary. Adaptive operations that can be triggered by the configuration manager are given in Figure 5c. The idea is, that if an FU is overloaded with work (for example the ALU

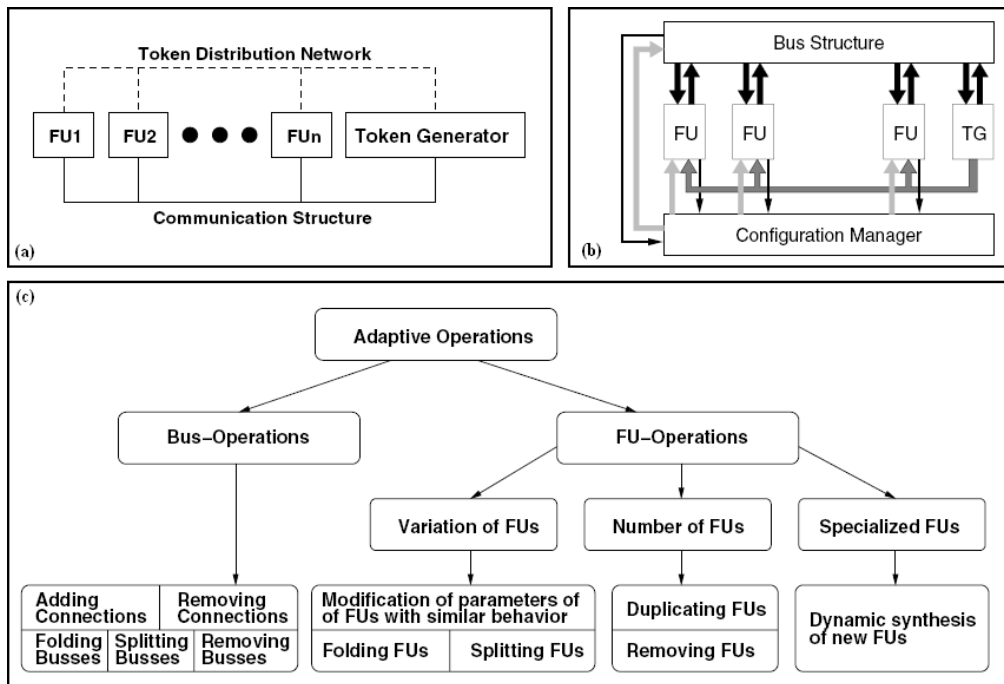


Figure 5: The fundamental design of AMIDAR (a) and its operation to organize (b) [9].

in the processor has to perform a lot of arithmetic operations and cannot keep up with incoming tokens), additional FU can fold into the actual work to assist. Similarly, if communication bandwidth becomes rare for an FU, additional communication lanes can be folded onto the one that reaches its maximum. For a complete overview on the functioning of the AMIDAR class of processor see [9].

Gatzka and Hochberger tested their architecture through deriving a Java-Bytecode processor from it. This processor was then used to execute two applications. The first application was written in a way, that it was data dominating (meaning that it produced high loads of data traffic, e.g., results by calculating the Integer Discrete Cosine Transformation). The second application was written in a way that it was control dominating (meaning it made excessive use of if, goto and other control statements). The idea was, that applications that focus on different tasks within a processor had to produce a different processor layout (controlled by the configuration manager). The tests showed, that a speedup of 27% was gained for application one and 16% for application two compared to a testrun where the self-organizing features of the processor were disabled. For additional results and information see [9].

The AMIDAR Class of processor is an example for self-organization. The basic concepts that are included in definition 2 (like increase in order, autonomy and adaptability) can be explained by the functionality of its architecture. The increase in order is performed through an adoption to the necessary functionality (in relation to the understanding of the word order as a layout to promote a certain function). FUs and the communication structure can be adjusted to the needs of the application that occupies the processor. The autonomy is given because no external controlling information is used by the processor to adjust its functionality (the configuration manager is part of the system itself).

4.2 Bio-Inspired Experiments in Robotics

Martinoli and Mondada describe an experiment that visualizes collective and group behaviours of robots [14]. The robots called Khepera (see Figure 6a) are 55mm in diameter. They have eight infrared sensors (Figure 6b), whereas 2 sensors are used to oversee the back of the robot. The robot is equipped with a grabber that can be lowered to grab things (Figure 6c). Two experiments were conducted using this type of robot, whereas we will take a closer look on only the first experiment. However, in both experiments, the robots did not communicate. They were unable to send or receive any information (except for the sensor gathered data). The first experiment dealt with collective noncooperative behaviour: robots had to perform tasks that could be performed without any cooperation between them. Experiment two dealt with collective cooperative behaviour: robots had to work together to accomplish a certain task.

To perform the first experiment, Martinoli and Mondada set up a field with random arrangements of robots and objects (Figure 6c). The robots were acting according to simple rules:

- Move around until the sensors detect an object.
- Grab the object.
- Again, move around until the sensors detect an object.
- Place the first object next to the recently discovered.
- Pull back and start at rule one.

While performing the actions permitted by the rules above, the robots used their sensors to avoid collisions with other robots. Depending on the number of robots used, the objects became clustered over time (see Figure 6d). Martinoli and Mondada found out, that using more than one robot has

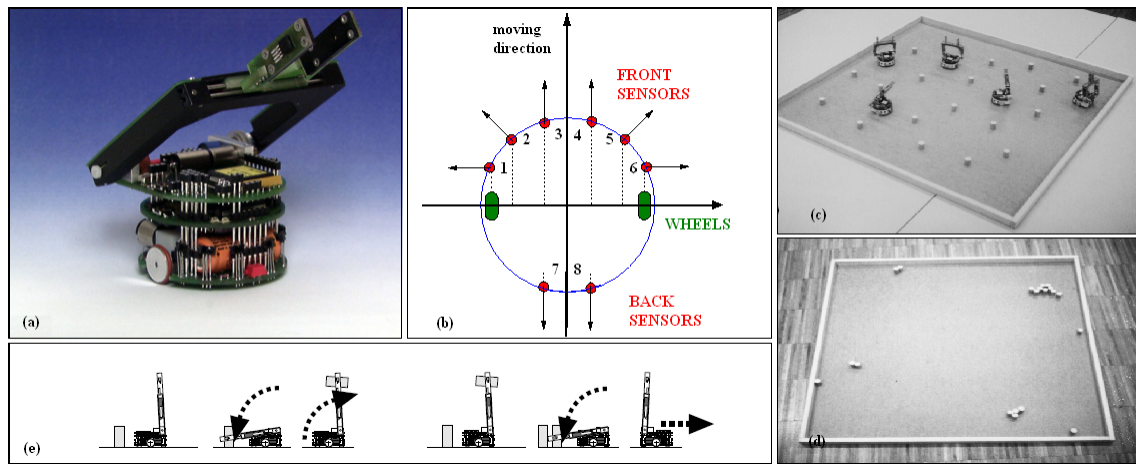


Figure 6: The Khepera robot (a) [23], its sensor layout (b), the experimental setup (c,d) and its movements (e) [14].

its advantages and disadvantages as well. For example, one single robot was only able to build small clusters (2 objects per cluster in average), while three robots built bigger clusters (3,5 objects in average). Contrariwise, a single robot dominated the average clustersize for most of the time, thus the advantage of multiple robots working together became visible only after some time passed by. Martinoli and Mondada discovered, that for their scenario with the field shown in Figure 6c, the optimal size for the number of robots is three. This is due to the fact that three robots reached the maximum of their average clustersize (3,5 objects per cluster) after ten minutes, while all other constellations took much longer.

This experiment is an example for emergence. The characteristics that are embedded in Definition 4 (like radical novelty, decentralization and robustness) can be described by the use of the property “average clustersize”. The property of “average clustersize” is obviously not applicable for a single object or robot. But with respect to the whole system, an average clustersize can be given (radical novelty). Decentralization is given, because the robots did not coordinate (neither driven through an internal or an external force of the system). Robustness can be visualized by moving around single objects. The average value for the clustersize will not dramatically change if an object is placed out of a cluster. Therefore the property of average clustersize degrades gracefully. The two-way link and the micro-macro effect become literally visible in this experiment, but it cannot be explained in detail why or where clusters will evolve only *that* they will evolve.

4.3 Artificial Antibodies

Many biological systems like us human beings use antibodies to defend themselves against intruders. Pietzowski et al. invented a system that uses the paradigm of antibodies to do the same for a middleware system called $OC\mu$ [19]. To fully understand the technical system, it is necessary to understand the basic functionality of the human immune system first.

The human immune system is aware of all objects that belong to the body itself. These objects are called “selves” [19]. There exists a special entity in our body, called the

Thymus, whose role it is to know all the selves. Antibodies that are created within our body have to somehow pass the Thymus *without* getting activated by it. If an antibody gets activated by the Thymus means, that this certain antibody would attack selfs within the body. These antibodies (and the B-Cells that create these antibodies) get singled out. The matter of activating/attacking is performed by the antibody through attaching itself to the proteine surface of some object (see Figure 7). This means, the Thymus is some sort of filter that singles out all antibodies that could attack selfs, but lets all antibodies pass that have no effects on selfs. Non-selfs get attacked by antibodies randomly (if some antibody exists within the body that has the needed proteine receptor). The human immune system is highly distributed besides the central entity of the Thymus [19]. The most important concepts here are *negative selection* and *receptors*. The negative selection is performed by filtering out all antibodies that have harmful proteine receptors and could attack selfs. Through evolution, the human immune system developed in a way, that it creates antibodies randomly and just singles out those, which could harm the body. Thereby the human immune system is able to learn and attack new non-selfs that appear over time. The receptors are used by antibodies to attach to objects within our body. Due to the nature of receptors, they only attach if an exact proteine-chain matches between the antibody and the object-surface is given. However, if an antibody has a short receptor (small proteine-chain), it can attach in the middle of a given object proteine-chain, thus it can check through the whole proteine-chain by testing all offsets from its beginning. Figure 9 visualizes an antibody with receptor-length 6 (grey rectangle) checking proteine-chains (at offset 2). The idea was used by Pietzowski et al. [19]. The technical system also distinguishes between selfs and non-selfs. In this case, selfs are messages that are being used by a middleware system (e.g. all messages the system can send and receive). The Thymus and its functionality was replaced by a certain service, called the Thymus Service (see Figure 8). The Thymus can ask all services within the middleware (and other middleware nodes) for their known messages and thereby knows all the “selfs” that are in use. The distribution of antibodies is not possible, as it is within the human body, therefore a

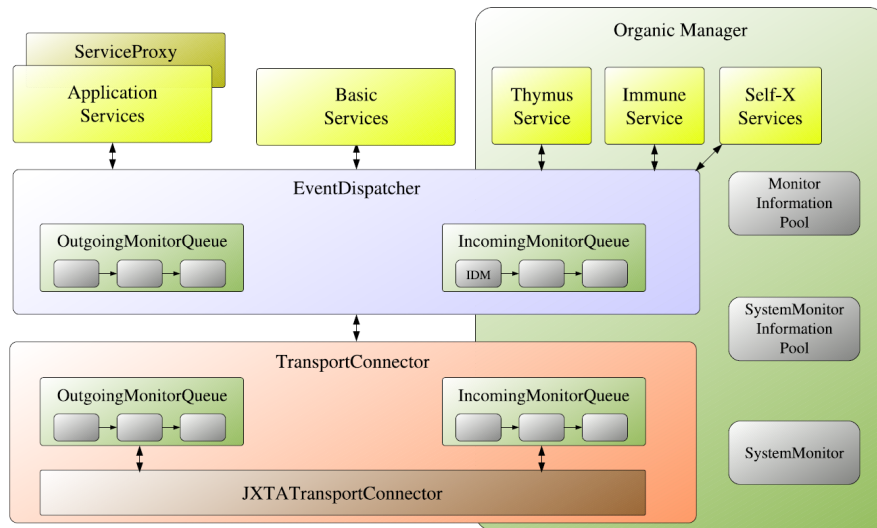


Figure 8: Architecture of the self-protecting middleware $OC\mu$ [19].

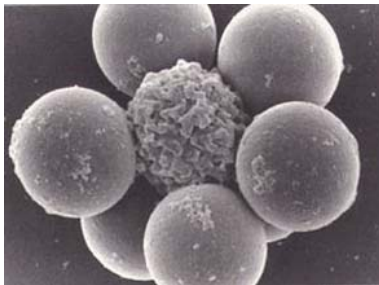


Figure 7: Antibodies (outer cells) attacking a non-self (inner cell).

Picture taken from www.nyu.edu/classes/ytchang/book/n005/antibody.jpg.

special filter component called Intrusion Detection Monitor (IDM), which checks all incoming messages, was designed. It is placed before the IncomingMonitorQueue (see Figure 8). The IDM communicates with the Immune Service, that acts like the antibodies and distinguishes between selfs and non-selfs. The Immune Service can block messages if it considers them malicious. Antibodies can be understood as bitchain-comparators. The creation and storage of antibodies is handled by the Immune Service. The Immune Service stores newly invented antibodies, if and only if the Thymus does not find a recognition conflict with a used system message, meaning the antibody does not match any offset in any self-message. Due to the complexity of antibody-creation and -storage, the process cannot be explained here in detail. The interested reader should see [19] for more information.

The results presented by Pietzowski et al. show, that a detection rate for non-selfs up to 99.3% is possible [19]. Furthermore, Pietzowski et al. determined the values that have to be known or adjusted to reach a detection rate this high. Among others, they are:

- the length of the self-messages,
- the receptor length of the antibodies,

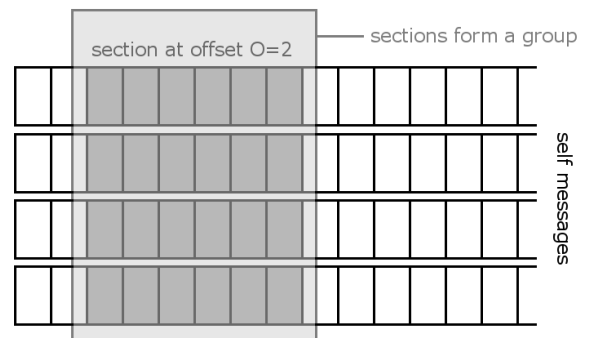


Figure 9: Proteine-chains checked with offset 2 [19].

- the amount of offsets to compare messages at,
- the amount of antibodies distributed into the system.

Since these are fixed values or values that can be computed dynamically, it is no problem to have $OC\mu$ defend itself in an efficient way. Furthermore, due to some storage optimizations, it was possible to reach this detection rate in a system with 1000 self-messages using only 9,77kb memory to store the antibodies. This is far less memory than it would need to store all the self-messages (15,6kb) to compare against. Concerning complexity-theory, the comparison of antibodies and self-messages is done in an efficient way. For more detailed information on the results see [19].

The $OC\mu$ middleware is an example for a self-protecting system, because no external information is necessary to the system. It needs to know what messages are in use and then creates its own antibodies to defend against malicious messages. It is also an example for self-organization, since its antibodies are adopted autonomously to new messages that become available to the system (controlled by the Thymus Service). It self-organizes in a way, that the optimal amount of antibodies is stored within the Immune Service, thereby maintaining order in its own “database”. Additionally, the

OC μ middleware is an example for emergence, since the system is able to defend itself not only against known attacks (as it is with virus scanners). It can detect new attacks and, due to the interaction of its service components, learns how to defend against them.

5. CONCLUSION

As we have seen, Organic Computing is a relatively new field of research, and thus it still lacks a coherent fundamental terminology. Scientists still debate on how to define the concepts of self-organization and other self-x properties as well as emergence. We have seen several definitions that I find promising, because they contain all important concepts that the paradigms of biological systems and societies can offer. Furthermore, new architectures like ADELFE and MASSIVE have been introduced as possible candidates to engineer OCSs. Examples for OCSs have been given, e.g. the AMIDAR class of processor and the OC μ middleware. Both systems implemented self-organizing features and proved their value to modern computer systems. Additionally, the bio-inspired experiment in robotics was discussed, that is supposed to visualize the concept of emergence. The benefit of emergents, e.g. the learning ability of OC μ , have also been presented. Though not directly productive yet, these examples illustrate the challenging vision of Organic Computing.

It is easy to invent yet another x-computing paradigm (like Organic-Computing or Ubiquitous-Computing) that envisions great new possibilities for future computer systems, but it is hard to augment these visions with practicable techniques to reach the envisioned goals [7]. OC is no different, but reasonable achievements have been made and were presented here.

6. ACKNOWLEDGMENTS

I want to thank Jörg Domaschka and Timo Ernst.

7. REFERENCES

- [1] Workgroup organic computing of the gi. *Position-Paper, Gesellschaft für Informatik*, <http://www.gi-ev.de/fileadmin/redaktion/Presse/VDE-ITG-GI-Positionspapier_20Organic_20Computing.pdf>, '03.
- [2] W. R. Ashby. Principles of self-organizing dynamic systems. *Journal of general psychology*, 37:125–128, 1947.
- [3] S. Auyang. Foundations of complex-system theories. *Cambridge University Press*, 1998.
- [4] C. Bernon, M. G. with S. Peyruqueou, and G. Picard. Adelfe, a methodology for adaptive multi-agent systems engineering. *ESAW 03*, (LNCS 2577):156–159, 2003.
- [5] O. Buchtala and B. Sick. Functional knowledge exchange within an intelligent distributed system. *ARCS 2007*, LNCS 4415:126–141, 2007.
- [6] K. Claffy and E. T. with D. McRobb. Internet tomography. *www.caida.org*, <<http://www.caida.org/publications/papers/1999/webmatters99/webmatters99.html>>, 1999.
- [7] J. Fromm. Ten questions about emergence. <<http://arxiv.org/abs/nlin/0509049v1>>, Sept. 2005.
- [8] J. Fromm. Types and forms of emergence. <<http://arxiv.org/abs/nlin/0506028v1>>, June 2005.
- [9] S. Gatzka and C. Hochberger. The organic features of the amidar class of processors. *ARCS 2005*, LNCS(3432):154–166, 2005.
- [10] J. Goldstein. Emergence as a construct: History and issues. *Emergence*, 1, 1999.
- [11] H. Kasinger and B. Bauer. Combining multi-agent-system methodologies for organic computing systems. *DEXA 05*, 05(1529-4188), 2005.
- [12] G. Lewes. *Problems of Life and Mind, Volume 2*. Kegan Paul, Trench, Turbner, London, 1875.
- [13] J. Lind. Iterative software engineering for multiagent systems: The massive method. *POLICY 2001*, Volume 1994/2001(ISSN 0302-9743), January 2001.
- [14] A. Martinoli and F. Mondada. Collective and cooperative group behaviours: Biologically inspired experiments in robotics. *ISER 95*, July 1995.
- [15] C. Müller-Schloer and B. Sick. Emergence in organic computing systems: Discussion of a controversial concept. *ITC 2006*, LNCS 4158:1–16, 2006.
- [16] C. Müller-Schoer. Organic computing - on the feasibility of controlled emergence. *CODES+ISSS'04*, (9):8–10, September 2004.
- [17] C. Müller-Schoer. Organic computing – systemforschung zwischen technik und naturwissenschaft. *it – information technology*, 47(4), Oldenbourg Verlag, 2005.
- [18] S. K. Mostefaoui and O. F. R. et. al. Self-organizing applications: A survey. *ESOA 03*, 2003.
- [19] A. Pietzowski, B. Satzger, W. Trumler, and T. Ungerer. A bio-inspired approach for self-protecting an organic middleware with artificial antibodies. *Lecture Notes in Computer Science, Springer Verlag Berlin/Heidelberg*, Volume 4124, September 2006.
- [20] I. Prigogine and P. Glansorff. *Thermodynamic study of structure, stability and fluctuations*. Wiley, 1978.
- [21] A. Rott. Self-healing in distributed network environments. *AINANW 07*, 2007.
- [22] H. Schmeck. Organic computing - a new vision for distributed embedded systems. *ISORC 05*, 2005.
- [23] L. Thomas. The department's new robot. *The Aberystwyth University, Computer Science Department Newspaper*, <<http://www.aber.ac.uk/compsci/Public/Recruitment/newsletter/Spring2001/>>, Spring 2001.
- [24] W. Trumler, F. Bagci, J. Petzold, and T. Ungerer. Towards an organic middleware for the smart doorplate project. *GI Jahrestagung 2004 (2)*, *GI*, pages 626–630, September 2004.
- [25] T. D. Wolf and T. Holvoet. Emergence versus self-organisation. *ESOA 2004*, LNCS 3465:1–15, Springer-Verlag, 2005.